Dissertation
zur Erlangung des akademischen Grades
Doctor philosophiae (Dr. phil.)

# Repräsentation, Verarbeitung und Organisation von Webinhalten: Relaunch des Webauftritts der Universität Bielefeld

eingereicht an der
Fakultät für Linguistik und Literaturwissenschaft
– Universität Bielefeld –

von Marc Kupietz

Juli 2003

Gutachter: Dr. Hans-Jürgen Eikmeyer
Prof. Dr. Gert Rickheit

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Beispielverzeichnis

# Einleitung

Eine durchdachte, funktionale Internetpräsenz wird nicht mehr nur von privatwirtschaftliche Unternehmen, sondern auch von öffentlichen Einrichtungen und insbesondere von Hochschulen selbstverständlich erwartet.

Die vorliegende Arbeit dokumentiert, wie eine solche Internetpräsenz mit dem Relaunch des Webauftritts der Universität Bielefeld umgesetzt wurde und beschreibt außerdem die zu dieser Realisierung notwendigen theoretischen Konzepte und Formalismen sowie deren Umsetzbarkeit in integrierten technischen Systemen. Ein wesentlicher Teil, der vom Autor geleisteten Arbeit, besteht dabei in der *Implementation* des dargestellten, an der Universität Bielefeld laufenden Systems.

Die für die Vereinheitlichung der im weiteren Sinne für das *World Wide Web* relevanten Formate und Techniken zuständigen Organisationen haben in den letzten Jahren eine große Anzahl an Standards, Empfehlungen und Entwürfen veröffentlicht. Ziel der vorliegenden Arbeit ist nicht zuletzt auch, eine Art roten Faden durch diese, inzwischen schwer überschaubare Menge zu ziehen und sowohl durch theoretische Begründung als auch durch die praktische Demonstration einen Weg aufzuzeigen, wie *Inhalte* am besten repräsentiert, organisiert und verarbeitet werden können. Der Fokus liegt dabei, ohne damit die Allgemeinheit über Gebühr einzuschränken, auf sogenannten *semistrukturierten* Inhalten, also solchen, die typischerweise *Texte* bzw. *Dokumente* ausmachen.

Im ersten Teil der Arbeit (Kapitel 1-3) werden, wie gesagt, zunächst die theoretischen Grundlagen zur Repräsentation von Inhalten und ihrer Aufbereitung für verschiedene Verwendungszwecke eingeführt. Kapitel 1 befasst sich insbesondere mit XML und flankierenden Standards sowie zugehöri-

gen Zugriffsschnittstellen. Das zentrale Thema dabei ist, wie Dokumente und Daten repräsentiert werden können ohne ihre spätere Verwendbarkeit einzuschränken. Dies wird zum einen theoretisch, anhand der Art der verwendeten Auszeichnungssprachen sowie der zugrundeliegenden Formalismen, zum anderen aber auch praktisch, anhand der *de-facto* vorhandenen Zugriffsmöglichkeiten diskutiert.

Im zweiten Kapitel werden dann Techniken vorgestellt, mit Hilfe derer sich solche *generischen*, also auf die Art ihres Inhalts spezialisierten, Auszeichnungssprachen wiederum in Sprachen mit allgemein festgelegter präsentationeller Semantik, wie z. B. HTML, übersetzt werden können.

Während in den ersten beiden Kapitel *Interoperabilität* im Wesentlichen auf rein *syntaktischer* Ebene diskutiert wird, ist *semantische Interoperabilität* in einer Extension des WWW, Thema des dritten Kapitels. Zum einen wird gezeigt, wie sich innerhalb des sogenannten *Semantic Web* allgemein und automatisch interpretierbare »Sprachen« ähnlich wie natürliche Sprachen entwickeln können, deren Ausdrucksmächtigkeit nicht, wie z. B. die von HTML, von vornherein eingeschränkt sind. Zum anderen wird gezeigt, wie sich ein *Semantic Web* auf Basis der zuvor vorgestellten Formalismen und Techniken *automatisch* mit semantisch transparenten Aussagen populieren lässt.

Im zweiten Teil der Arbeit (Kapitel 4) wird dargestellt, wie sich die in den vorausgegangenen Kapiteln eher unzusammenhängend vorgestellten Formalismen und Techniken in konkreten Softwaresystemen, sogenannten *Content Management Systemen*, verbinden und integrieren lassen. Darüber hinaus wird gezeigt, wie sich bereits thematisierte wichtige Grundprinzipien der Contenthaltung, insbesondere *Interoperabilität* und *Redundanzvermeidung*, auch in Bezug auf die Organisationen und Verwaltung von Inhalten innerhalb solcher Systeme verwirklichen lassen. Eine weitere Funktion des vierten Kapitels besteht darin, die für Kapitel 5 benötigten Konzepte des *Content Managements* einzuführen.

Im letzten Teil der Arbeit (Kapitel 5) wird schließlich der Relaunch des Webauftritts der Universität Bielefeld dokumentiert und dabei demonstriert, wie sich die in den ersten beiden Teilen der Arbeit aufgestellten Prinzipien und vorgestellten Techniken im Kontext einer Hochschule in die Praxis umsetzen lassen.

# 1 Repräsentation von Inhalten: Formate und Schnittstellen

Ziel dieses Kapitels ist zunächst, die grundlegenden Datenformate, Schnittstellen und Techniken einzuführen, auf deren Basis insbesondere für das *Web* bestimmte Inhalte erstellt, repräsentiert, abgefragt und verändert werden können. Bei dieser Einführung soll außerdem bereits besonderes Augenmerk auf die jeweiligen Vorzüge und relevanten Nachteile eines Formalismus in Bezug auf seinen Verwendungszweck gelegt werden. Technische Details werden demzufolge insbesondere dann eine Rolle spielen, wenn sie in Bezug auf diese Diskussion oder ein grundlegendes Verständnis relevant sind.

## 1.1 XML und angrenzende Standards

Die *eXtensible Markup Language* (XML) [XML00; MBK+00][1] wurde als Vereinfachung der bereits 1986 als ISO-Standard registrierten *Standard Generalized Markup Language* (ISO-8879) vom W3-Konsortium[2] (W3C) entwickelt und 1998 als Standard verabschiedet. SGML und XML werden als sogenannte *Meta-Markupsprachen* klassifiziert, d. h. als Sprachen auf deren Basis Markupsprachen (deutsch: *Auszeichnungsprachen*) definiert und realisiert werden können. Die bekanntesten Vertreter solcher Auszeichnungsprachen sind HTML – eine SGML-Anwendung – und XHTML – eine XML-Anwendung.

[1] Die hier zitierte Version stellt eine Überarbeitung der ersten als Standard verabschiedeten Version dar.

[2] `http://www.w3.org`

```
<xsd:simpleType name="kleineZahl">
  <xsd:restriction base="xsd:positiveInteger">
  <xsd:maxInclusive value="100"/>
  </xsd:restriction>
</xsd:simpleType>
```

**Beispiel 1.1.1:** *Datentypdefinition in XML-Schema.*
Ausgehend von dem nativen XML-Schema-Datentyp `positiveInteger` wird der Datentyp `kleineZahl` definiert, der den Wertebereich auf positive Zahlen bis einschließlich 100 eingrenzt.

### 1.1.1 DTDs und Schemasprachen

Die Struktur von SGML- und XML-Sprachen kann in sogenannten *Document Type Definitions* (DTDs) festgelegt werden. In DTDs wird die Reihenfolge und die hierarchischen Abhängigkeiten der für den Dokumenttyp relevanten Textteile (*Elemente*) definiert und damit die *Tags* für die später zu erstellenden Dokumente dieses Typs festgelegt. Den Elementen können in der DTD zusätzlich *Attribute* zugeordnet werden, welche Eigenschaften der Elemente repräsentieren. Während SGML-Anwendungen grundsätzlich aus einer DTD und dieser entsprechenden Dokumenten, sogenannten Instanzen, bestehen, sind DTDs bei XML optional. Zusätzlich zum Status der *Validität* (oder *Gültigkeit*) für korrekte Instanzen einer DTD, wurde dazu der Status der *Wohlgeformtheit* eingeführt, den solche Dokumente innehaben, die auf keine DTD referieren, aber den allgemeinen syntaktischen XML-Metaregeln (es muss genau ein Wurzelelement existieren, Starttags und Endtags müssen ausbalanciert sein, Elemente müssen ohne Überlappungen ineinander verschachtelt sein, ...) genügen.

In Nachfolge des alten DTD-Standards wurden in jüngster Zeit weitere Formalismen zur Definition von XML-Dokumentklassen entwickelt. Wichtigster Vertreter dieser sogenannten *Schemasprachen*, deren Gemeinsamkeit unter anderem darin besteht, dass diese (im Gegensatz zu DTDs) wiederum selbst XML-Anwendungen sind, ist das vom W3C 2001 als Empfehlung erlassene *XML-Schema* (XSD) [XML01a; Wal02]. XML-Schema erlaubt unter anderem durch die Einführung eines hierarchischen Datentypkonzepts deutlich feinere syntaktische Restriktionen. So lässt sich z. B. der genaue Aufbau von terminalen Zeichenketten durch XML-Schema festlegen (siehe Beispiel 1.1.1),

während DTDs keine weitere Klassifikation von *Character Data* erlauben und damit z. B. nicht einmal eine Unterscheidung von Zahlen und Text zulassen. Die Unterstützung von *Namensräumen* (s.u.) in XML-Schema sorgt außerdem für eine bessere Wiederverwendbarkeit von Dokumentgrammatiken und Teilen von Dokumentgrammatiken.

### 1.1.2 Namensräume

Um XML-Dokumenten zu erlauben, das Markupvokabular mehrerer XML-Sprachen zu verwenden, ohne dass dabei Namenskollisionen entstehen oder die Information, zu welcher Sprache ein bestimmtes Element oder Attribut gehört, verloren geht, wurden vom W3C sogenannte *XML-Namensräume* (»*XML namespaces*«) [XML99; MBK+00] eingeführt. Die Zugehörigkeit eines Elements oder eines Attributs zu einem bestimmten Namensraum, wird dabei durch das Voranstellen einer durch einen Doppelpunkt abgetrennten Namensraumabkürzung (*Proxy*) markiert. Diese Abkürzung muss zuvor in einer *Namensraumdeklaration* dem eigentlichen *Namensraumnamen*, der durch einen eindeutigen URI repräsentiert wird, zugeordnet werden (siehe Beispiel 1.1.2 ).

Das den Namensräumen zugrundeliegende Ziel ist, einerseits bestehende XML-Sprachen möglichst wiederverwendbar zu machen und andererseits Dokumente gleichzeitig für verschiedene Anwendungszwecke auszeichnen und strukturieren zu können. Beide Ziele werden allerdings nur mit gewissen Einschränkungen erreicht. Zum einen sind Namensräume nicht Bestandteil des weitaus älteren DTD-Standards, so dass sich eine gemischte Wiederverwendung von XML-Sprachen, die noch durch DTDs definiert sind, ohne weiteres nur in Bezug auf diese verarbeitende Anwendungen, nicht aber in Bezug auf ihre Dokumentgrammatiken realisieren lässt. Zum anderen sind der multiplen Strukturierbarkeit von XML-Dokumenten durch Wohlgeformtheitsregeln Grenzen gesetzt, so dass z. B. zwei Elemente, selbst wenn sie zwei verschiedenen Namensräumen zugehörig sind, sich nicht überlappen dürfen.[3]

[3] Siehe [Wit01] für eine ausführliche Diskussion zur multiplen Strukturierbarkeit von XML-Dokumenten.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xhtml
  xmlns:upm="http://www.uni-bielefeld.de/pressemitteilungen"
  xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Pressemitteilungen</title></head>
  <body>
    <upm:mitteilungen>
      <table>
        <upm:pressemitteilung>
          <tr>
            <td>
              <upm:nr>27/2003</upm:nr><br/>
              <upm:datum>13.02.2003</upm:datum>
            </td>
            <td>
              <upm:titlehead>
                Universitätsgesellschaft vergibt Preis für gute Lehre
              </upm:titlehead>
              <br/>
              <upm:title>
                Studenten sind aufgerufen Kanditaten zu benennen
              </upm:title>
            </td>
          </tr>
        </upm:pressemitteilung>
      </table>
    </upm:mitteilungen>
  </body>
</xhtml>
```

**Beispiel 1.1.2:** *Deklaration und Verwendung von XML-Namensräumen.*
Im Beispiel werden die Namensräume `http://www.uni-bielefeld.de/pressemitteilungen` (Präfix »upm«) und `http://www.w3.org/1999/xhtml` (Defaultnamensraum – kein Präfix) deklariert und verwendet. Je nach intendiertem Verwendungszweck, z. B. *Anzeige im Webbrowser* oder *Weiterverarbeitung*, ist die Strukturierung des Dokuments nach Elementen des Default- bzw. des `upm`-Namensraums relevant.

### 1.1.3 Adressierung von Ressourcen

Noch grundlegender für das *World Wide Web* als eine universelle Sprache zur Repräsentation strukturierter Dokumente ist eine einheitliche und universelle Konvention um *Ressourcen* (z. B. Dokumente, Grafiken, Musikdateien, usw.) eindeutig bezeichnen zu können und damit eindeutig auf sie referieren zu können. Ohne ein solches einheitliches Adressierungsschema wäre eine gemeinsame Nutzung von Ressourcen und insbesondere eine *Verlinkung* zwischen Ressourcen oder Ressourcenfragmenten nicht möglich.

Ursprünglich wurden im WWW sogenannte *Uniform Resource Locators* (URLs) zur Adressierung von Ressourcen verwendet. URLs sind in der Regel (es gibt auch andere Varianten) folgendermaßen aufgebaut:

```
<Schema>:<Adresse der Resource>
```

`Schema` steht dabei für den Namen des Protokolls (z. B. `http`, `ftp`, `mailto`, `news`, `imp`, usw.) mit dem auf die Ressource zugegriffen werden kann. Abhängig vom Adressierungsschema handelt es sich bei der Adresse z. B. um einen Pfad im Dateisystem oder eine E-Mail-Adresse. Ebenfalls abhängig vom Adressierungsschema kann dem URL separiert durch ein »#« ein sogenannter *Fragment Identifier* hinzugefügt werden. Seine Bedeutung hängt dabei von der Art der Ressource ab. Bei HTML-Ressourcen wird durch den *Fragment Identifier* z. B. auf ein Element mit einem entsprechenden ID-Attribut referiert.

Über die Bezeichnung von real im Web existenten Entitäten hinaus entstand schnell die Notwendigkeit, auch solche Entitäten bezeichnen zu können, die nicht im Web lokalisierbar sind. Zu diesem Zweck wurde der URL, bevor seine Funktion überhaupt exakt festgelegt war, durch den allgemeineren *Uniform Ressource Identifier* (URI) ersetzt. Nach der allgemeineren Definition des URI kann eine Ressource alles sein, was eine Identität hat, z. B. also auch eine Person, ein Film oder eine Radiosendung oder auch eine Menge solcher Ressourcen. Eine *Ressource* wird dabei nicht als eine bestimmte Entität zu einem bestimmten Zeitpunkt, sondern als *konzeptuelle Abbildung* (»*conceptual mapping*« [BLFM98]) auf eine Entität oder eine Menge von Entitäten verstanden, so dass eine Ressource konstant bleiben kann, selbst wenn

sich ihr Inhalt – also die zur ihr korrespondierenden Entitäten – ändert, solange dabei die konzeptuelle Abbildung erhalten bleibt. Praktisch heißt das, dass ein URI eine Entität auch *attributiv* [vgl. Don66] kennzeichnen kann, in dem Sinne, dass man mit ihm z. B. auf den *aktuellen deutschen Bundeskanzler* oder *die aktuelle Version von Microsoft Word* verweisen kann, ohne sich dabei auf einen bestimmten Bundeskanzler oder eine bestimmte MS-Word-Version festzulegen.

Einen weiteren Standard zur Adressierung von Ressourcen und eine Unterklasse der URIs bilden sogenannte *Uniform Resource Names* (URNs) [Moa97]. Diese wurden ursprünglich als Gegenstück zu URLs entwickelt um Ressourcen mit persistenten, permanenten Namen zu versehen, die im Gegensatz zur URLs nicht mit der Lokation von Ressourcen, sondern mit ihrem Inhalt variieren. Dieser Gegensatz hat sich allerdings inzwischen durch die allgemeine Verwendung von URIs zur persistenten Adressierung auch abstrakter Ressourcen und die in der Praxis übliche Gleichsetzung von URLs und URIs größtenteils aufgehoben. Nach heutiger Sichtweise bildet der URN-Präfix `urn:` ebenso wie z. B. `http:` lediglich ein URI-Schema, mit Hilfe dessen nicht die Zugriffsmethode, sondern spezielle Namensräume definiert werden. Z. B. ergibt die Menge aller URIs der Form `urn:isbn:n-nn-nnnnnn-n` einen URN-Namensraum [MD02].

### 1.1.4 Generisches vs. visuelles Markup

*Markup* ist ein Begriff aus dem Verlagswesen und bezeichnete zunächst die Markierungen oder Annotation eines Autors oder Typographen, wie eine Passage im Text layoutet oder gedruckt werden sollte. Während diese Instruktionen früher von Setzern bzw. Layoutern umgesetzt wurden, konnte der Vorgang später durch die Ersetzung von Schreibmaschine durch Computer automatisiert werden. Auch bei den heute gängigen Textverarbeitungssystemen hat sich an der Vorgehensweise lediglich geändert, dass Markupbefehle bzw. Steuerzeichen in der Regel nicht mehr auf dem Bildschirm dargestellt werden, sondern Attribute wie z. B. Fettdruck direkt dargestellt werden. Geblieben ist das Prinzip, dass das Verfassen eines Textes mit der Festlegung seines späteren Aussehens fest verknüpft ist. Um dem Leser das Erkennen

```
<section>
 <title>Meta-Auszeichnungssprachen</title>
 <para>
   Die <foreignphrase lang="en">eXtensible Markup
   Language</foreignphrase>
   <acronym>XML<acronym>) <citation>XML1.0</citation> ist eine
   <firstterm>Meta-Auszeichnungssprache</firstterm>.
```

```
<vertical-space length="1cm"/>
<font size="14pt" weight="bold"
            family="sans-serif">Meta-Auszeichnungssprachen</font><br/>
<span align="both">Die
  <italic>eXtensible Markup Language</italic>
 (XML) [Bray et al. 2000] ist eine
 <italic>Meta-Auszeichnungssprache</italic>.
```

**Beispiel 1.1.3:** *Generische bzw. visuelle Auszeichnung mit XML.*
Wie die Beispiele zeigen, ist XML unabhängig von der Art eines verwendeten Markups. D.h. XML-Sprachen können z. B. sowohl generische (oben) als auch darstellungsorientierte Auszeichnungen (unten) vorsehen. Streng genommen tragen zwar die Tagnamen wie `font` oder `title`*selbst* keine Bedeutung (siehe dazu Abschnitt 3.1). Entscheidend ist jedoch, dass aus dem `italic`-Tag nicht eindeutig rekonstruiert werden kann, was dieser ursprünglich aussagen sollte, da er einmal zur Auszeichnung einer englischsprachigen Passage und einmal zur Auszeichnung eines neu eingeführten Begriffs verwendet wird.

der intendierten Struktur eines Textes zu vereinfachen, muss der Verfasser diese *sichtbar* machen, indem er z. B. seine Abschnittsüberschriften mit dem Attribut »fett« markiert. Die Vorteile dieses sogenannten *visuellen Markups*, allgemeiner auch *prozedurales Markup*, liegen vor allem in der geringen Anforderung an das Textverarbeitungssystem und in der schnellen Umsetzbarkeit individueller Layoutvorstellungen.

Wie bereits zu Anfang erwähnt, handelt es sich bei XML nicht um eine einfache Markupsprache, sondern um eine *Metasprache*, in der sich – eingeschränkt durch einige syntaktischen Restriktionen – beliebige Auszeichnungssprachen realisieren lassen. Konkret bedeutet dies, dass XML-Anwendungen an keine bestimmte *Art* von Markup, also insbesondere auch nicht an visuelles Markup, gebunden sind.

Das für XML-Sprachen typische, sogenannte *deskriptive* oder *generische Markup* zeichnet sich im Gegensatz zum prozeduralem oder visuellem Markup

dadurch aus, dass ein Autor Textpassagen nicht mit Informationen über deren intendiertes Aussehen versieht, sondern mit *generischen* Informationen, d. h. mit Informationen bezüglich ihrer *Art* bzw. *Rolle* oder *Funktion*. Darüber hinaus erlaubt XML die *Strukturierung* von Dokumenten: zum einen durch die generische Auszeichnung von Elementen eines Dokuments und zum anderen durch die Beziehung, in die die Elemente zueinander gesetzt werden.

Ein großer Teil der Vorteile eines generischen Markups erwächst aus der Tatsache, dass ein Autor beim Verfassen eines Dokuments nicht gezwungen ist, sein Wissen über Rolle und Art der Textelemente und deren Strukturierung in eine bestimmte optische Repräsentation zu transformieren. Bei einer solchen Transformation gehen zwangsläufig Informationen verloren. Z. B. kann aus einer als »fett« markierten Textpassage nicht eindeutig rekonstruiert werden, dass diese Textpassage als Abschnittsüberschrift gedacht war. Demzufolge kann eine Textverarbeitung im Nachhinein weder z. B. eine Abschnittsnummer berechnen noch automatisch einen Eintrag im Inhaltsverzeichnis erzeugen. Der durch ein reines visuelles Markup bedingte Verlust ursprünglich vorhandener generischer Informationen schränkt die *Verwendbarkeit* eines Dokuments auf eine rein visuelle Darstellung ein. Dabei ist außerdem durch die Vermischung von inhaltlichen und gestalterischen Informationen auch die Art und Weise der Darstellung bereits im Wesentlichen festgelegt und im Nachhinein nur schwer modifizierbar.

XML dagegen erlaubt eine Trennung von inhaltlichen und gestalterischen Informationen und eine Auszeichnung von Inhalten durch prinzipiell *alle* vorhandenen Zusatzinformationen. Die Verwendbarkeit und spätere Wiederverwendbarkeit und damit letztlich die Nachhaltigkeit eines XML-Dokuments ist also durch den XML-Formalismus selbst nicht eingeschränkt.

Eine geeignete Auszeichnung und geeignete Transformationswerkzeuge vorausgesetzt (siehe dazu Kapitel 2), können so verschiedene *Ansichten* – sogenannte *Views* – auf ein Dokument erzeugt werden. Denkbar sind z. B. mediumspezifisch optimierte *Views*: zur Anzeige im Web, zum Vorlesen durch einen Sprachsynthetisator, zum Druck als Buch (Stichwort *Single Source Publishing* – siehe Abschnitt 2.5); oder auch zweckspezifische *Views*, z. B. für unterschiedliche Nutzergruppen.

### 1.1.5 Syntaktische Interoperabilität

XML und *generisches Markup* sind, wie oben bereits angedeutet, voneinander unabhängig. Das heißt zum einen, dass Daten oder Dokumente auch ohne XML generisch ausgezeichnet werden können und zum anderen, dass ein wohlgeformtes XML-Dokument nicht notwendigerweise auch generisch ausgezeichnet ist. Letzterer Umstand soll an dieser Stelle vor allem deshalb noch einmal erwähnt sein, da *XML* als Schlagwort häufig fälschlicherweise mit »*gutem*«, d. h. generischem Markup gleichgesetzt wird (siehe Beispiel 1.1.3 auf Seite 9).

Ein weiterer Grund für die Verwendung von XML zur Auszeichnung von *Content* – unabhängig von der Art der Auszeichnung – ist allerdings der (nicht grundlos) hohe Grad der Verbreitung von XML und XML-Applikationen.

Verantwortlich für die extrem schnell erreichte Popularität der *eXtensible Markup Language* waren im Wesentlichen zwei Faktoren: ihre universelle Verwendbarkeit auf der einen und ihre Einfachheit auf der anderen Seite. Die universelle Verwendbarkeit beruht dabei zum einen darauf, dass XML als *öffentlicher* Standard vom W3-Konsortium – der derzeitigen Autorität für Webstandards schlechthin – erlassen wurde. Dadurch ist der XML-Standard frei verwendbar und sein Weiterbestehen ohne Abhängigkeiten von Softwareherstellern und deren Firmenpolitik gewährleistet. Zum anderen beruht die universelle Verwendbarkeit auf der mit seiner Eigenschaft als Meta-Auszeichnungssprache einhergehenden Erweiterbarkeit bzw. der Nicht-Festlegung auf ein bestimmtes Lexikon und ausreichenden Möglichkeiten zur Strukturierung von Inhalten. Durch Unicode-Unterstützung ist XML außerdem international verwendbar, während durch Namensräume Benennungskonflikte und strukturelle Ambiguitäten vermieden werden können und mittels DTDs oder Schemata die Syntax von XML-Sprachen verbindlich und überprüfbar ist. XML ist also ausreichend mächtig und fein granulierbar, um als eine Art universelle *Meta-Lingua-Franca* dienen zu können. Über dieses Potential hinaus hat sich XML aufgrund der immer breiter werdenden Softwareunterstützung auch von großen kommerziellen Anbietern wie Sun oder Microsoft inzwischen zu einem – auf der syntaktischen Ebene – *tatsächlich*

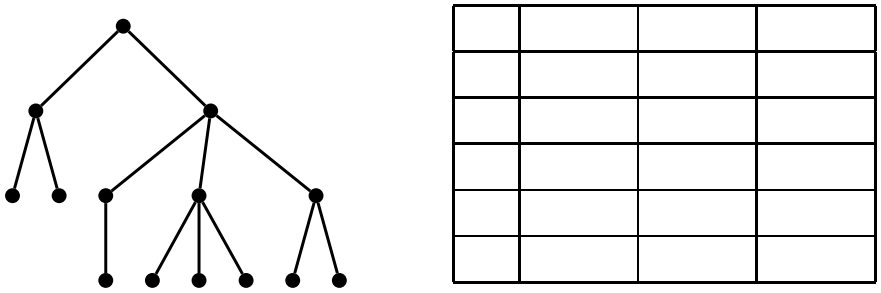

**Abbildung 1.1:** Typische Struktur von XML-Dokumenten bzw. Datenbanktabellen

interoperablen Standardformat entwickelt.

## 1.2 XML im Kontext relationaler Datenbanken

Aufgrund der, in den vorausgegangenen Abschnitten thematisierten, Universalität und Interoperabilität und der leichten Transportabilität von XML-Dokumenten sind XML-Sprachen als Austauschformat zwischen Applikationen unangefochten. Zur nativen Repräsentation von Quelldaten stellen jedoch klassische relationale Datenbanksysteme (RDBMS) unter Umständen eine Alternative zu XML dar.

Im Folgenden sollen deshalb zunächst einige prinzipielle Faktoren bezüglich der Art und Struktur von Daten diskutiert werden, anhand derer ein geeignetes Repräsentationsmedium gewählt werden kann.

### 1.2.1 Tabellen, Bäume und Graphen

Der augenfälligste Unterschied zwischen relationalen Datenbanken und XML-Dokumenten liegt in ihrem jeweils typischen Aufbau. Während XML-Dokumente eine baumartige, typischerweise stark verzweigende, hierarchische Struktur aufweisen, bestehen Datenbanken grundsätzlich aus Tabellen (siehe Abbildung 1.1).

Folglich legen Daten mit stark hierarchischer Struktur, wie z. B. Textdoku-

mente, die aus Kapiteln, die wiederum aus Abschnitten, die wiederum aus Absätzen usw. bestehen eher eine XML-Repräsentation nahe, während leicht tabellarisch darstellbare Daten eine Speicherung in Datenbanken nahelegen. Trotz der Augenfälligkeit der typischen Strukturunterschiede ist dieses Kriterium nicht unbedingt entscheidend. Auf der einen Seite lassen sich Tabellen leicht als Bäume darstellen und auf der anderen Seite sind Bäume prinzipiell auch in Tabellen überführbar. Bei letzterem Weg gehen allerdings die Effizienzvorteile von RDBMS unter Umständen verloren. Des Weiteren ist die Relevanz des Kriteriums dadurch eingeschränkt, dass Daten und Dokumente nur selten auf eine Menge perfekter Bäume (also auf einen *Wald*) abbildbar sind. Um allgemeinere Graphen in XML zu repräsentieren, sind – ähnlich wie in Tabellen bereits für die Repräsentation von Bäumen – Zeiger notwendig, die Dokumentfragmente miteinander verknüpfen. Wenn zu viele dieser Zeiger nötig sind, ist unter Umständen wieder eine tabellarische Darstellung vorzuziehen [vgl. Bax02].

### 1.2.2 Semistrukturierte Daten

Eine weitere naheliegende und eher informelle Klassifikation, die für die Auswahl des Repräsentationsmediums relevant ist, ist die Unterscheidung zwischen eher *dokumentorientierten* und *datenorientierten* (bzw. *data- vs. document-centric* [Bou02]) Inhalten. In typischen Fällen ist diese Klassifikation wiederum leicht zu treffen: Bei Romanen oder Gedichten handelt es sich eher um Dokumente, während es sich z. B. bei Fahrplaneinträgen einer Buslinie um typische Daten handelt.

Eine genauere Klassifikation von Inhalten lässt sich nach *Art* und *Grad* der *Strukturiertheit* treffen. XML-Auszeichnungen eignen sich dabei besonders für sogenannte *semistrukturierte* Daten. Semistrukturierte Daten sind dabei durch folgende Merkmale charakterisiert [vgl. ABS00]:

- *Heterogene Elementstruktur:* Die Struktur der Elemente, aus denen sich die Daten zusammensetzen, ist uneinheitlich. Während einigen Elementen bestimmte Informationen fehlen, enthalten andere Elemente zusätzliche Informationen bzw. Attribute. Die Inhalte gleicher Elemente können außerdem unterschiedlichen Typs sein. `Preis`-Elemente kön-

nen zum Beispiel teilweise Euro- und teilweise DM-Beträge enthalten. Außerdem kann der Grad der Unterstrukturierung variieren, während z. B. einige `publisher`-Elemente nur eine Zeichenkette enthalten, können andere z. B. weiter in `publishername` und `address` untergliedert sein.

- *Nur partielle Strukturierbarkeit:* Teile der Daten, z. B. Pixelgrafiken, sollen unter Umständen nicht weiter strukturiert werden, andere Teile, z. B. Text nur stellenweise.

- *Indikative im Gegensatz zu normativer Struktur:* Während in Standard-Datenbankanwendungen eine strenge Typisierung der Werte in Tabellenspalten erwünscht und notwendig ist, um die Integrität der Daten und die Funktionsfähigkeit der Applikationen zu gewährleisten, kann diese Einschränkung bei anderen Anwendungen zu restriktiv sein. In diesen Fällen ist es sinnvoll, Datentypen (zunächst) nur im Sinne einer Richtlinie und nicht als strenge Restriktion zu spezifizieren (siehe auch *Heterogene Elementstruktur* oben).

- *A-posteriori-Richtlinien im Gegensatz zu A-priori-Schemata:* Während in Standard-Datenbankanwendungen die Struktur der Datenbank *a priori* in sogenannten *Schemata* festgelegt werden muss, ist es in Fällen in denen die Inhalte nicht exakt vorhersagbar sind, notwendig, die Spezifikation der Datenrichtlinien mit der Art der tatsächlich auftauchenden Daten evolvieren zu lassen.

- *sehr große Schemata:* Aufgrund heterogener Elementstruktur (s.o.) und häufiger optionaler Argumente würde das für eine Datenbank zu spezifizierende Schema so extreme Ausmaße annehmen, dass die Leerstellen in den Datenbanktabellen deutlich mehr Platz einnehmen als die eigentlichen Daten.

- *schnell evolvierende Struktur:* In Standard-Datenbanksystemen ist die Anpassung von Schemata an veränderte Datenstrukturen mit hohen Kosten verbunden. In vielen Anwendung müssen aber Strukturänderungen aufgrund neuer Daten gemacht werden.

- *veränderliche Anforderungen:* Bei der Sammlung von Daten sind die Anforderungen an das speichernde System häufig – durch verschiedene Phasen bedingte – Veränderungen unterworfen. In einer ersten Phase sollen z. B. Daten oder Textpassagen nur unstrukturiert gesammelt werden und erst in den darauf folgenden Phasen nach und nach mit Strukturinformationen versehen werden. In der letzten Phase erst, soll die Struktur dann für Recherche- oder Präsentationszwecke optimiert sein.

- *variable oder uneinheitliche Anforderungen:* Denkbar ist auch, dass eine Daten- oder Textbasis gleichzeitig unterschiedlichen Anforderungen genügen muss und für jede dieser Anforderungen eine individuelle Strukturierung notwendig ist oder nur spezielle Strukturinformationen relevant sind. Ein solcher Fall ist z. B. gegeben, wenn für unterschiedliche Benutzergruppen spezielle *Views* (siehe Abschnitt 1.1.4) vorgesehen sind.

Diese Charakterisierung semistrukturierter Daten ist gleichzeitig eine Aufzählung und Motivation der Eigenschaften und Möglichkeiten, die XML-Anwendungen von Standard-Datenbankanwendungen abheben.

### 1.2.3 Vorläufige Schlussfolgerungen

Es wurde gezeigt, dass sich nicht vollständig und persistent strukturierte Daten besser innerhalb eines XML-Frameworks repräsentieren lassen. Im Hinblick auf eine aus verarbeitungsökonomischen Gründen wünschenswerte uniforme Behandlung *aller* Arten von Inhalten scheint sich also *XML* als grundlegender Formalismus anzubieten.

Allerdings kann die Eignung von XML- bzw. herkömmlichen Datenbanklösungen natürlich nicht allein anhand der jeweilig zugrundeliegenden Datenmodelle entschieden werden. Vielmehr ist es notwendig, diese im Zusammenhang mit den Zugriffssystemen zu vergleichen, in die sie eingebettet sind, da nicht die Datenmodelle, sondern Zugriffsmethoden umfassende Systeme letztendlich dafür ausschlaggebend sind, wie die Daten verarbeitet, abgefragt oder verändert werden können. Aus diesem Grund soll die vergle-

chende Diskussion bei der Vorstellung von Anfrage- und Manipulationssprachen für XML-Daten in den folgenden beiden Abschnitten fortgesetzt werden und in Abschnitt 1.5 auf tatsächliche Implementationen solcher Schnittstellen innerhalb sogenannter XML-Datenbanken bezogen werden.

## 1.3 XML-Anfragesprachen

Auch detailliert ausgezeichnete und gut strukturierte Datenbestände sind wertlos, wenn keine geeigneten Methoden existieren um bestimmte Daten aufzufinden und abzufragen. Im Datenbankbereich existieren zum Auffinden von Informationen inzwischen sehr ausgereifte *Anfragesprachen*. Die wichtigsten Vertreter sind SQL [siehe z.B. Dat87], für relationale, und OQL für objektorientierte Datenbanken [siehe z.B. Heu97].

### 1.3.1 XPath

Die älteste und wichtigste XML-Anfragesprache ist XPath [Kay00]. XPath ist essentieller Bestandteil von XSLT (siehe Abschnitt 2.3) und nimmt dort in etwa die gleiche Rolle ein wie `SELECT`-Ausdrücke innerhalb SQL. Trotz seiner engen Verbundenheit mit XSLT ist XPath, da es im Prinzip auch separat nutzbar ist, in einer eigenen W3C-Empfehlung definiert [XPa99]. Der Aufbau von XPath-Anfragen entspricht typischerweise einem Pfad von einem bestimmten Knoten aus zu einem oder mehreren Zielknoten. Ähnlich wie Pfade in einem Verzeichnisbaum sind dabei die einzelnen Pfadelemente, bzw. Knoten, durch '/' voneinander getrennt, wobei ein '/' am Anfang eines Pfades auch auf den Wurzelknoten referiert. Zusätzlich können hinter jedem Knoten auf dem Pfad in eckige Klammern, ähnlich wie in SQL `WHERE`-Klauseln, Bedingungen angegeben werden. Das Ergebnis eines solchen Anfragepfades ist eine – eventuell leere – Menge von Knoten.[4] Der Ausdruck:

```
/events/event[department='Fakultät für Physik']/title
```

[4] Genauer gesagt ist das Ergebnis in der aktuellen XPath-Version 1.0 vom Typ *`result-tree-fragment`*. Siehe dazu auch Abschnitt 2.3.2.2

könnte z.B., angewendet auf ein XML-Dokument, das eine Liste von Veranstaltungen beinhaltet, die Titelknoten aller Veranstaltungen der Fakultät für Physik liefern.

Die Pfadausdrücke in XPath erlauben allerdings nicht nur das schrittweise Durchlaufen des XML-Baums entlang vorhandener Kanten in der Richtung »Wurzel nach Blatt«. Für jeden Schritt in einem Pfad (ein sogenannter *Step*) kann zusätzlich durch einen Achsenspezifikator (`AxisSpecifier`) festgelegt werden, in Richtung welcher Achse der Schritt erfolgen soll. Durch den Achsenspezifikator `ancestor::` geht der Schritt z. B. in Richtung der Vorgänger des aktuellen Knotens, während z. B. durch `preceding-siblings::` die Achse der Geschwister des aktuellen Knotens, die vor diesem bzw. links von ihm liegen, ausgewählt wird. Falls keine Achse angegeben ist, wird, wie im Beispiel oben, die `child::`-Achse, also die direkten Nachfolger des aktuellen Knotens, gewählt.

Eine weitere wichtige Eigenschaft von XPath ist, dass nicht nur Knoten des aktuellen Dokuments, sondern mittels der `document(URI)`-Funktion auch auf beliebig viele andere XML-Dokumente zugegriffen werden kann. Dabei kann es sich bei dem `URI`-Parameter um einen berechneten Wert handeln, für den außerdem nicht nur der Typ `String`, sondern auch eine Menge von Strings zugelassen ist.[5] Das Resultat der Funktion `document(node-set)` ist dann die Vereinigung aller Wurzelknoten, der durch die in `node-set` angegebenen XML-Dokumente. Auf diese Weise kann in XPath auf eine vorher nicht bekannte Menge von Dokumenten zugegriffen werden. Außerdem ist es so mit XPath möglich, Semi-Verbünde[6] (*Semi-Joins*) und mit Hilfe von XSLT -entsprechend den Fähigkeiten von SQL- auch allgemeine Theta-Verbünde (*Theta-Joins*) zu erzeugen.

[5] Genauer gesagt ist der Typ `node-set`, eine Menge von Knoten, von denen jeder `text`-Wert in einen URI-bezeichnenden `String` umgewandelt wird.

[6] Semi-Verbünde sind Verbünde zweier Relationen (bzw. XML-Dokumente oder Datenbanken) `R` und `S` mit dem Selektionsprädikat $\Theta$ deren Ergebnis lediglich aus Tupeln einer der beiden Relationen besteht. Die andere Relation wird nur für $\Theta$ verwendet [LL95].

```
SELECT veranstaltung.titel, einrichtung.titel
 FROM veranstaltung, einrichtung
 WHERE veranstalung.einrichtung = einrichtung.id;
```

```
<xsl:for-each select="document('veranstaltungen.xml')
                    /veranstaltung">
  <xsl:variable name="id" select="einrichtung"/>
  <title>
     <xsl:value-of select="titel"/>
  </title>
  <veranstalter>
     <xsl:value-of select="document('einrichtungen.xml')
                    /einrichtungen[id=$id]/titel"/>
  </veranstalter>
</xsl:for-each>
```

**Beispiel 1.3.1:** *Konstruktion eines Verbunds mit SQL bzw. XSLT/XPath.*
Die Beispiellistings (SQL oben, XSLT/XPath unten) erzeugen (entsprechende Datenbasen vorausgesetzt) jeweils eine Liste mit den Titeln aller Veranstaltungen und deren ausrichtende Einrichtungen.

#### 1.3.1.1 XPath-Probleme

Zwar ist es möglich, mit XPath, zuständig für den Datenzugriff und XSLT, zuständig für die Datenkonstruktion, die meisten Funktionalitäten von SQL nachzubilden, jedoch ergeben sich dabei einige Schwierigkeiten.

Ein unerwünschter Effekt ist, dass durch die meist notwendige Interaktion von XPath und XSLT, Anfragen nicht nur komplizierter werden, sondern sie auch ihre Deklarativität verlieren. Wie Beispiel 1.3.1 zeigt, ist z. B. für die Konstruktion eines Verbunds ein Schleifenausdruck notwendig.

Außerdem verfügt XPath selbst über keine Funktionen zur Sortierung, Maximum-, Minimum- oder Durchschnittsbildung und erlaubt auch nicht die Definition eigener Funktionen. Viele für datenzentrierte Anwendungen typische Berechnungen müssen so bei jeder Abfrage umständlich und schwer lesbar nachgebildet werden. Aber auch bei dokumentorientierten Anwendung kann es zu Problemen kommen. Beispiel 1.3.2 zeigt eine Funktion, die den Namen der aktuellsten Datei liefert, die vom aktuellen Dokument eingebunden wird. Da XPath über keine Maximumfunktion verfügt, muss diese über den Umweg einer durch XSLT vorgenommenen Sortierung nachgebildet werden.

```
<xsl:for-each select="//include/@file">
  <xsl:sort select="document(.)//modification-date"
            order="descending"/>
  <xsl:if test="position() = 1">
    <xsl:value-of select="."/>
  </xsl:if>
</xsl:for-each>
```

**Beispiel 1.3.2:** Suche nach der neusten eingebundenen Datei innerhalb eines Dokuments

### 1.3.2 XQuery und XQueryX

Um den Anforderungen an eine dedizierte Datenanfragesprache sowohl für datenorientierte als auch für dokumentorientierte Inhalte besser gerecht zu werden, wird vom W3C der XQuery-Standard entwickelt (zur Zeit noch im Entwurfsstadium Stadium [XQu01; MSC+02]). XQuery ist eine Weiterentwicklung der Abfragesprache *Quilt* [CRF00], die wiederum grundlegend auf Syntax und Ideen von SQL und OQL und den ersten XML-Anfragesprachen XQL, XML-QL und schließlich XPath 1.0 basiert.

Ebenso wie die kommende XSLT-Version 2.0 [XSL01b] wird XQuery die kommende XPath-Version 2.0 [XPa02] beinhalten. XPath 2.0 verfügt gegenüber XPath 1.0 im Wesentlichen über folgende Erweiterungen:

- Quantoren
- Schnitt- und Differenzmengen-Funktionen
- konditionale Ausdrücke
- mächtigere Pfadausdrücke
- neue Stringfunktionen (insbesondere Funktionen zu Ersetzen)
- neue Aggregationsfunktionen (Maximum, Minimum, ...)
- reguläre Ausdrücke
- Unterstützung für XML-Schema-Datentypen [XML01b]

Ein großer Teil der gegenüber SQL vermissten Funktionalität wird in Zukunft also bereits durch XPath realisiert sein.

```
for $b in document("veranstaltungen.xml")/veranstaltung,
    $a in document("einrichtungen.xml")/einrichtung
where $b/einrichtung = $a/id
return
     <title>
       { $b/title }
     </title>
     <veranstalter>
       { $a/titel }
     </veranstalter>
```

**Beispiel 1.3.3:** Konstruktion eines Verbunds mit XQuery

Das Herzstück von XQuery sind die sogenannten FLWR (sprich *flower*) Ausdrücke, die Iterationen und Variablenbindungen für Zwischenergebnisse ermöglichen. Diese Art von Ausdrücken ist z. B. nützlich für die Konstruktion von Verbünden von mehreren Dokumenten und die Restrukturierung von Daten. FLWR steht für die Schlüsselwörter `for`, `let`, `where` und `return`, entsprechend der Basisstruktur der FLWR-Ausdrücke:

for *`variable bindings`* where *`condition`* return result

In der `for`-Klausel wird dabei die Bindung von Variablen über alle Werte ihres jeweiligen Ausdrucks in Dokumentreihenfolge iteriert. In der `let`-Klausel werden ebenfalls Variablen gebunden, allerdings ohne Iteration. Die so generierten Bindungstupel werden in der optionalen `where`-Klausel gefiltert, so dass die `return`-Klausel nur für Tupel ausgeführt wird, die die Bedingung der `where`-Klausel erfüllen.

Ebenso wie XSLT erlaubt XQuery natürlich auch die Konstruktion neuer Elemente. Beispiel 1.3.3 zeigt wie in XQuery ein restrukturierter Verbund aus Veranstaltungs- und Einrichtungsdaten erzeugt werden kann. Beim Vergleich mit den entsprechenden Anfragen in XSLT/XPath in Beispiel 1.3.1 auf Seite 18 sieht man, dass mit der XQuery-Lösung eine starke Vereinfachung gewonnen ist.

Des Weiteren fällt auf, dass die XQuery-Anfrage selbst, im Gegensatz zum XSLT/ XPath-Pendant, nicht XML-konform ist. Die Begründung ist, dass mit XQuery eine in erster Linie von Menschen gut les- und schreibbare Anfragesprache geschaffen werden soll. Mit *XQueryX* existiert aber eine XQuery genau entsprechende Anfragesprache mit XML-Syntax. Diese XML-Version

hat trotz ihrer vermeintlichen Unlesbarkeit (siehe Beispiel 1.3.4 ) den Vorteil, mit Standard-XML-Tools verarbeitet werden zu können. Außerdem ist es denkbar, per XQuery(X) wiederum Anfragen an XQueryX-Anfragen zu stellen, XQuery-Anfragen per XSLT zu transformieren und zu erzeugen und Anfragen in XML-Dokumente einzubinden.

## 1.4 Manipulation von XML-Bäumen

Ein schwerwiegender Nachteil von XML-Systemen ist, dass zur Zeit noch kein zu SQL äquivalenter Standard zur Veränderungen von Daten in XML-Dokumenten, auch `UPSERT`-Standard[7] genannt, besteht. Selbst der in der Entwicklung befindliche XQuery-Standard – das zukünftige XML-Pendant zu SQL aus der Datenbankwelt – unterstützt bisher keine, für die Entwicklung von XML zu einem Standard-Repräsentationsformat aber unerlässliche, Datenmodifikation. Wie bei XSLT ist auch bei XQuery das Quelldokument strikt vom Zieldokument getrennt, in dem Sinne, dass aus dem Quelldokument ein Zieldokument generiert wird, nicht aber das Quelldokument selbst verändert wird.

Aufgrund des fehlenden `UPSERT`-Standards müssen zur Modifikation von XML-kodierten Daten in der Regel komplette Dateien bearbeitet werden. Die Möglichkeit, zusammengehörige Daten und deren Auszeichnung insgesamt bearbeiten zu können, ist zwar häufig ein erwünschtes Feature, das Fehlen von feiner granulierten Alternativen kann jedoch in einigen Fällen eine Reihe von Nachteilen mit sich bringen:

- Unnötig große Datenmengen müssen zum Laden und Speichern transportiert werden.
- Änderungen an nicht intendierten Stellen sind durch Applikationen schwer auszuschließen.
- Die Wahrung von Gültigkeit und Wohlgeformtheit muss über komplette Dokumente überprüft werden.

[7] `UPSERT` ist eine Verschmelzung aus den SQL-Befehlen zum Verändern und Einfügen einer Tabellenzeile: `UPDATE` und `INSERT`.

```
<query xmlns="http://www.w3.org/2001/06/xqueryx">
  <flwr>
    <forAssignment variable="$b">
      <step axis="CHILD">
        <function name="document">
          <constant datatype="CHARSTRING">veranstaltungen</constant>
        </function>
        <identifier>veranstaltung</identifier>
      </step>
    </forAssignment>
    <forAssignment variable="$a">
      <step axis="CHILD">
        <function name="document">
          <constant datatype="CHARSTRING">einrichtungen</constant>
        </function>
        <identifier>einrichtung</identifier>
      </step>
    </forAssignment>
    <where>
      <function name="EQUALS">
        <step axis="CHILD">
          <variable>$b</variable> <identifier>einrichtung</identifier>
        </step>
        <step axis="CHILD">
          <variable>$a</variable> <identifier>id</identifier>
        </step>
      </function>
    </where>
    <return>
      <elementConstructor>
        <tagName><identifier>titel</identifier></tagName>
        <step axis="CHILD">
          <variable>$b</variable> <identifier>titel</identifier>
        </step>
      </elementConstructor>
      <elementConstructor>
        <tagName><identifier>veranstalter</identifier></tagName>
        <step axis="CHILD">
          <variable>$a</variable> <identifier>titel</identifier>
        </step>
      </elementConstructor>
    </return>
  </flwr>
</query>
```

**Beispiel 1.3.4:** Konstruktion eines Verbunds mit XQueryX

- Während Änderungen vorgenommen werden, muss das gesamte Dokument für andere Schreibzugriffe gesperrt sein.
- Zur Veränderung von XML-Daten muss auf XML-Editoren

  zurückgegriffen werden, deren Bedienungsaufwand unter Umständen den zu machenden Veränderungen nicht angemessen ist.

Um diese Nachteile und Einschränkungen zu umgehen, existieren zur Zeit zwei Möglichkeiten:

Die erste besteht darin, die Manipulation von XML-Dokumenten in Allzweck-Programmiersprachen zu implementieren. Mit dem *Document Object Model* (DOM) existiert für viele Programmiersprachen zwar ein standardisiertes API zur Manipulation von XML-Bäumen (siehe dazu Abschnitt 2.2), jedoch ist die *Beschreibungsebene* dieses API deutlich niedriger als z. B. die von SQL. Das heißt zum einen, dass der Implementationsaufwand zur Nachbildung von aus SQL bekannten UPSERT-Funktionalitäten recht hoch ist und zum anderen, dass ein Großteil des Codes abhängig von einer speziellen XML-Anwendung ist und nicht für andere Anwendungen wiederverwendet werden kann.

Die zweite Möglichkeit besteht darin, auf im Entwicklungsstadium befindliche XML-UPSERT-Sprachen zu setzen. Den Entwurf einer solchen Sprache stellt das von der XML:DB-Initiative entwickelte *XUpdate* [LM00] dar. Bei XUpdate handelt es sich um eine XML-Anwendung, d. h. Dokumentmanipulationen werden selbst wiederum als wohlgeformte XML-Dokumente ausgedrückt. Zur Verarbeitung und Erzeugung von XUpdate-Dokumenten können so vorhandene XML-Werkzeuge, wie z. B. XSLT, verwendet werden. Der Nachteil der XML-basierten Syntax ist allerdings, dass im Vergleich zu SQL manuelle Datenmanipulationen recht komplex sind. Im Gegensatz zu XQuery schlägt XUpdate keine für manuelle Zugriffe spezialisierte Syntaxvariante vor.

Der dritte und vermutlich beste Weg besteht daher darin, die zukünftige Standardanfragesprache XQuery um Manipulationsoperationen zu erweitern, um so zum einen XQuery-Funktionalitäten nutzen zu können und zum anderen auf den bereits definierten Syntaxvarianten konsistent aufsetzen zu

können. Entsprechende Vorschläge existieren bereits seit einiger Zeit [siehe TH01; Leh01].

## 1.5 XML-Datenbanksysteme

XML-Datenbanken sollen die Dichotomie zwischen reinen XML-Anwendungen und reinen RDBMS auflösen, indem sie XMLs flexible Strukturierbarkeit mit den effizienten Zugriffs- und flexiblen Kontrollfunktionalitäten herkömmlicher Datenbanksysteme unter einer Oberfläche vereinen. Unter dem Begriff *XML-Datenbanken* sind also komplette *Systeme* zu verstehen, die nicht nur die eigentliche Speicherung von Daten leisten, sondern auch Methoden zur Zugriffskontrolle und Schnittstellen zur Speicherung, Abfrage und Manipulation von Daten zur Verfügung stellen. Aufgrund eines noch nicht konsolidierten Marktes,[8] unterschiedlicher Zielsetzungen, unterschiedlicher Funktionsweise[9] und nicht zuletzt auch wegen der Abwesenheit eines *High-Level-Standards* zur Manipulation von XML-Daten (siehe Abschnitt 1.4) unterscheiden sich diese Schnittstellen allerdings deutlich stärker voneinander als die konventioneller SQL-Datenbanksysteme.

Die Gemeinsamkeiten zwischen den Systemen bestehen zumeist in der Implementation von Mechanismen zur Zugriffskontrolle, zum Transaktions-Management sowie der Implementation einer XPath-Schnittstelle und der *low-level-APIs* DOM und SAX (siehe Abschnitt 2.2). Außerdem hat sich das Konzept sogenannter *Collections* durchgesetzt, in denen eine ausgewählte Menge von Dokumenten zusammengefasst behandelt, also z. B. durchsucht werden können. Die Anfragesprache XQuery ist bereits in einigen Sys-

[8] Allerdings zeichnet seit einiger Zeit eine Vormachtstellung der von der Software AG entwickelten XML-Serversoftware *Tamino* (`http://www.tamino.com`) von der Software AG ab.

[9] Man unterscheidet bezüglich der Funktionsweise vor allem zwischen sogenannten *XML Enabled Databases* (XED), die XML-Daten-Zugriffe typischerweise nicht vollständig strukturerhaltend auf relationale DBMS abbilden und sogenannten *Nativen XML Datenbanken* (NXD), die Daten in speziell auf XML zugeschnittenen Datenstrukturen speichern, vollständig strukturerhaltend arbeiten und nur XML-spezifische Zugriffsarten (z. B. XPath) erlauben.

Eine häufig aktualisierte Übersicht über Datenbanksysteme verschiedener Hersteller findet sich auf den Seiten von Roland Bourret `http://www.rpbourret.com/xml/XMLDatabaseProds.htm`.

temen implementiert, so dass nach einer Veröffentlichung von XQuery als W3C-Standard mit einer sehr breiten Unterstützung zu rechnen ist. Weniger optimistisch ist die Situation im Hinblick auf Manipulationssprachen einzuschätzen. In diesem Bereich herrschen anbieterspezifische Lösungen vor, wobei weder eine allgemeine Verwendung XUpdate[10] noch die Herausbildung sonstiger De-facto-Standards in naher Zukunft in Aussicht steht. Wahrscheinlich ist zur Zeit die Herausbildung einer Interoperabilitätsschicht auf niedriger Ebene. So wird eine ebenso wie XUpdate von XML:DB-Initiative entwickelte, speziell auf XML-Datenbanken zugeschnittenes Java-API (XAPI[11]), das Datenmanipulation sowohl auf DOM- als auch auf Textbasis vorsieht, bereits von vielen Anbietern unterstützt. Möglicherweise können auf Basis dieser Schnittstelle zukünftige Zugriffssprachen-Standards anbieterunabhängig implementiert werden, so dass XML-Datenbanken eine ähnlich hohe (Inter-)Operabilität erreichen wie klassische SQL-Datenbanken.

Eine mögliche Parallelentwicklung ist, dass sich aufgrund der sich abzeichnenden Vormachtstellung von *Tamino* und der Vorteile des Ansatzes (s.o.), die von *Tamino* verwendete, auf XQuery aufgesetzte Manipulationssprache [siehe Ges03] als Standard durchsetzen wird.

## 1.6 Benutzerschnittstellen

### 1.6.1 Praktische Probleme

Ein großes praktisches Problem der generischen Auszeichnung von semistrukturierten Inhalten ist, dass die damit verbundene besondere Art des Editierens von Dokumenten von vielen Anwendern nicht akzeptiert wird.

Die Gründe dafür sind vielschichtig. Zum einen wird der Sinn einer generischen Auszeichnung häufig nicht verstanden, so dass folglich auch der Umgang mit in spitze Klammern eingeschlossenen »Steuerbefehlen« gegenüber »modernen« WYSIWYG-Editoren nicht als Fortschritt, sondern als ei-

[10] XUpdate ist zwar in einer der wichtigsten Open-Source-XML-Datenbanklösungen, dem von Apache entwickleten *XIndice* (`http://xml.apache.org/xindice`) implementiert, jedoch ist aufgrund der oben angesprochenen Designmängel eine darüber hinaus gehende Etablierung unwahrscheinlich.

[11] `http://www.xmldb.org/xapi/`

ne Art Rückschritt »*in die Zeit der Bernsteinmonitore*« aufgefasst wird. Sowohl Teil als auch Ursache des fehlenden Verständnis ist dabei, dass »XML« häufig fälschlicherweise mit generischem Markup gleichgesetzt wird. Die Konvertierbarkeit eines Word-Dokuments nach »XML«, bzw. in ein »*gutes*« generisches Auszeichnungsformat, wird deshalb folgerichtig auch nicht als ein Problem der Auszeichnung aufgefasst, sondern als »*vermutlich bereits gelöstes, rein technisches Problem*«, für dessen Lösung man »*nur geeignete Software benötigt*«.

Die Verortung der Verantwortlichkeit bei der Textverarbeitungssoftware ist dabei auch nicht völlig von der Hand zu weisen. Jedoch basieren XML-Export-Plugins für Textverarbeitungsünd DTP-Software auf Formatvorlagen und können deshalb auch nur dann bei akzeptablem manuellen Aufwand akzeptable Ergebnisse erzielen, wenn dem Benutzer das Konzept generischer Auszeichnung bekannt ist und er Formatvorlagen entsprechend konsequent im generischen Sinne und nicht im Hinblick auf ihre assoziierte präsentationelle Semantik verwendet. Letzteres erfordert allerdings in Text- und DTP-Programmen vermutlich mehr Disziplin als in speziellen XML-Editoren.

Ein weiteres Problem besteht darin, dass Standard-Textverarbeitungssoftware nur eine einfache Aneinanderreihung von Formatelementen, aber keine hierarchische Verschachtelung dieser erlaubt. Um die fehlende Strukturierbarkeit zu kompensieren und damit die eindeutige Abbildbarkeit eines »flachen« Dokuments auf ein gültiges XML-Dokument zu gewährleisten, müssen deshalb Formatelemente sehr stark ausspezifiziert werden. Z. B. müsste der Titel eines Abschnitts von vornherein anders ausgezeichnet werden als der Titel eines Unterabschnitts, um Unterabschnitte von Abschnitten unterscheidbar zu machen. Die Alternative zu einer solchen Überspezifikation von Elementen ist in der Regel die manuelle Entscheidung struktureller Ambiguitäten im Verlauf des Exports bzw. der Konversion. Wenn jedoch nicht mehr alle notwendigen Informationen im Dokument selbst enthalten sind, sondern bei jeder Konversion manuell hinzugefügt werden müssen, ist eine Haltung des Contents in der Textverarbeitung kaum noch sinnvoll.

Selbst wenn Autoren das Konzept und die prinzipielle Zweckmäßigkeit generischer Auszeichnungen verstehen, folgt daraus jedoch keineswegs eine ausreichende Motivation und Akzeptanz. Häufig liegt das daran, dass Auto-

ren es gewohnt sind, ihre Texte nur für ein bestimmtes Ausgabemedium zu produzieren, nämlich ihren eigenen Drucker oder ihren eigenen Webbrowser. Falls eine andere Verwendungsart benötigt wird, werden die damit verbundenen Konversionsarbeiten nicht von den Autoren selbst, sondern von »Computerexperten«, Webmastern oder Verlagen durchgeführt. Eine vom Präsentationsmedium unabhängige Auszeichnung bedeutet deshalb, selbst wenn die Reduktion des Gesamtaufwands vielleicht unstrittig ist, für die Autoren selbst zunächst nur zusätzlichen Arbeitsaufwand. Hinzu kommt, dass Autoren häufig die Einschränkung auf generisches Markup als Einschränkung ihrer gestalterischen Kreativität empfinden.

Andererseits sind Akzeptanzprobleme natürlich nicht nur auf die mangelnde Einsicht der Autoren zurückzuführen. Ein generelles Dilemma ist sicherlich, dass naturgemäß gerade verbreitete XML-Dokumentgrammatiken (z. B. TEI[12] oder Docbook[13]) umfangreicher und komplexer sind als dies für jede einzelne konkrete Anwendung an sich notwendig wäre. Der Einarbeitungsaufwand für Autoren zur Beherrschung der vorhandenen Elemente und Attribute, sowie deren Kombinierbarkeit und Strukturierbarkeit, steht damit häufig in keiner Relation zur eigentlichen Aufgabe. Voraussichtlich wird sich dieses Dilemma jedoch mit der Zeit durch die Verbreitung von XML-Schema und die dadurch ermöglichte bessere Modularisierbarkeit von Dokumentgrammatiken und Wiederverwendbarkeit von Dokumentgrammatik-Modulen in verschiedenen Kontexten allmählich auflösen.

### 1.6.2 XML-Editoren

Während spezielle XML-Editoren sicher kaum dazu geeignet sind, das Verständnis *generischen Markups* zu verbessern, können sie zumindest helfen, den zur Bearbeitung von XML-Dokumenten notwendigen Aufwand zu reduzieren und damit die Akzeptanz einer XML-konformen Auszeichnung zu verbessern. Derzeit gebräuchliche XML-Editoren wie z. B. XMetaL[14] von Co-

[12] `http://www.tei-c.org/`
[13] `http://www.docbook.org/`
[14] `http://www.xmetal.com`

rel, XMLSpy[15] und die frei erhältlichen Alternativen JEdit[16] mit XML-Plugin und GNU Emacs[17] bzw. XEmacs[18], jeweils im PSGML-Modus[19], bieten eine ganze Reihe von Funktionen zur Vereinfachung des Editierens von XML-Dokumenten.

Grundlegende Features sind z. B. automatische Validierung mit detaillierten Hinweisen auf Fehler, (halb-)automatische Schließen offener Elemente, DTD-sensitives Einfügen von Elementen und Attributen, »Einklappen« von Elementen zur Verbesserung der Übersichtlichkeit sowie automatisches Einrücken entsprechend der Verschachtelungstiefe. Besonders *XMetal* bietet zudem einige Features, die den Umstieg von herkömmlichen Textverarbeitungssystemen erleichtern. So können z. B. Tags vollständig ausgeblendet werden und Dokumente so mit *Cascading Stylesheets* (siehe Abschnitt 2.1) verknüpft werden, dass das direkte grafische Feedback beim Bearbeiten eines Dokuments dem eines herkömmlichen WYSIWYG-Editors wenig nachsteht. Für die Eingabe eher datenorientierter Dokumente sind dagegen insbesondere tabellarische Ansichten und die Einschränkung von Werten durch die Unterstützung von XML-Schema hilfreich.

### 1.6.3 Webformulare: XForms

Trotz oder auch gerade wegen der zahlreichen Hilfsfunktionen erfordern XML-Editoren nachwievor einen gewissen Einarbeitungsaufwand und stellen damit für die gelegentliche Eingabe und Veränderungen von Daten unter Umständen für den Benutzer eine unangemessen hohe Hürde dar. Gerade für die Bearbeitung stark strukturierter Daten sind daher, ähnlich wie im Umgang mit Datenbanken üblich, auf spezielle Anwendungen zugeschnittene Webinterfaces anwendungsunspezifischen XML-Editoren vorzuziehen. Das Problem bei solchen Webinterfaces besteht allerdings darin, dass ihre Implementation und anwendungsspezifische Anpassung mit deutlichem höherem Programmieraufwand verbunden ist als vergleichbare Implementationen für

[15] `http://www.xmlspy.com`
[16] `http://www.jedit.org`
[17] `http://www.gnu.org/software/emacs/`
[18] `http://www.xemacs.org`
[19] `http//sourceforge.net/projects/psgml`

Datenbankanwendungen, so dass die »Hürde« im Wesentlichen nur auf die Seite der Programmierung verschoben wird.[20] Einen Ausweg aus der jetzigen Situation verspricht der sich zur Zeit in der Entwicklung befindliche *XForms-Standard* [XFo02].

XForms werden in zukünftigen XHTML-Standards die bisher in HTML üblichen Formulare ersetzen.[21] Gegenüber den von HTML oder XHTML 1.0 her bekannten Formularen ermöglichen XForms insbesondere – in Analogie zur Trennung von Inhalt und Form – eine *Trennung von Zweck und Form*. Das heißt, um z. B. vom Benutzer eine von mehreren Optionen auswählen zu lassen, muss nicht explizit eine Gruppe von sogenannten Radioknöpfen angegeben werden, sondern zunächst nur der *Zweck* der Dateneingabe-Aufforderung, nämlich eine Auswahl aus mehreren Optionen. Wie diese Auswahl vom *User Agent* dargestellt ist, ist abhängig vom Aus-/ Eingabemedium und kann über *Cascading Stylesheets* beeinflusst werden.

Neben einigen weiteren »Modernisierungen« wie z. B. einer XML-konformen Internationalisierung und der bereits angesprochenen Medienunabhängigkeit ist das interessanteste neue Feature von XForms, dass ausgefüllte Formulare nicht mehr standardmäßig als unstrukturierte Attribut-Werte-Paare an den Server zurückgeliefert werden, sondern im XML-Format. Wie diese XML-Rückgabe strukturiert sein soll und welchen Beschränkungen sie genügen soll, kann dabei in externen XML-Schema-Beschreibungen (siehe Abschnitt 1.1.1) und internen Schema-Spezifikationen festgelegt werden. Auch die vorgegebenen Instanzen der Schemata können sowohl innerhalb des Dokuments als auch durch Referenzen auf externe XML-Dokumente bestimmt werden. Die Bindung zwischen den Eingabe-/ Interaktionselementen und den zugehörigen Knoten im XML-Instanz-Baum werden dabei durch XPath-Ausdrücke hergestellt.

[20] Zu einem gewissen Teil verantwortlich für den größeren Programmieraufwand ist sicherlich auch der fehlende UPSERT-Standard (siehe Abschnitt 1.4). Im Folgenden wird allerdings weniger die Kommunikation zwischen Applikation und XML-Datenbasis als die Kommunikation zwischen Applikation und Anwender thematisiert.

[21] XForms ist bereits Bestandteil des im Entwurfsstadium befindlichen XHTML2.0-Standards [XHT02].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html>
  <head>
    <xfm:model>
      <xfm:instance xlink:href="tl.xml"/>.
      <xfm:submitInfo id="submit1" method2="postxml"
                      to="www.example.org"/>
    </xfm:model>
  </head>
  <body>
    <table border="">
      <tr>
        <th>Name</th>
        <th>Semester</th>
        <th>e-mail</th>
      </tr>
      <xfm:repeat ref="/tnl/tn" id="repeatTn">
        <tr>
          <td><xfm:input ref="name"/></td>
          <td><xfm:input ref="semester"/></td>
          <td><xfm:input ref="email"/></td>
        </tr>
      </xfm:repeat>
    </table>
    <xfm:button id="insert">
      <xfm:caption>Einfügen</xfm:caption>
      <xfm:insert ref="/tnl/tn" at="xfm:cursor('repeatTn')"
                  position="after" ev:event="click" />
    </xfm:button>
    <xfm:button id="delete">
      <xfm:caption>Löschen</xfm:caption>
      <xfm:delete ref="/tnl/tn" at="xfm:cursor('repeatTn')"
                  ev:event="click" />
    </xfm:button>
    <xfm:submit name="Submit" ref="test" to="submit1">
      <xfm:hint>Speichern</xfm:hint>
      <xfm:caption>Save</xfm:caption>
    </xfm:submit>
  </body>
</html>
```

**Beispiel 1.6.1:** XForms-Formular zum Editieren einer Seminarteilnehmerliste

#### 1.6.3.1 Benutzerinterface-Programmierung

Diese Eigenschaften von XForms werden es in Zukunft, wenn XForms ausreichend von Browsern unterstützt wird,[22] ermöglichen, mit sehr wenig Aufwand Eingabemasken zur Modifikation von XML-Daten zu erzeugen. Insbesondere zum Editieren von stark strukturierten Daten könnte auf den Einsatz von XML-Editoren verzichtet werden. Daraus ergäbe sich nicht nur ein quantitativer Vorteil in Bezug auf Kosten und Schulungsaufwand, sondern ein grundsätzlicher qualitativer Unterschied. Dieser liegt darin begründet, dass durch die Verwendung von XForms die Schnittstelle zwischen der XML-Applikation (z. B. einem *Web-Content-Management- System* – siehe Kapitel 4) und dem Editiervorgang nicht mehr auf den Austausch kompletter Dokumente beschränkt wäre, sondern durch die XML-Applikation frei bestimmt werden könnte[23]. Das heißt nicht nur, dass auch ausgewählte Fragmente eines Dokuments ausgetauscht werden könnten, sondern dass auch der clientseitige Prozess des Editierens (z. B. Darstellung der Formulare, Validitätsüberprüfung, Hilfstexte und Aktionen) indirekt unter der Kontrolle der XML-Applikation läge. Diese Art der Kontrolle konnte bisher nur durch aufwendige und stark browserspezifische Javascript-Programmierung erreicht werden.

Im Gegensatz zum imperativen Ansatz von Javascript erlaubt XForms aber insbesondere eine weitestgehend abstrakte deklarative Formulierung der Interaktionsparameter. So muss für die Überprüfung von Eingaben auf Korrektheit und die Benutzerhinweise auf fehlerhafte Eingaben in XForms lediglich auf ein – in den meisten Fällen ohnehin vorhandenes – XML-Schema verwiesen werden, während diese Funktionalität bei herkömmlichen HTML-Formularen für jedes Formular und jede Änderung aufs Neue in Javascript neu programmiert werden muss. XForms könnte einem ewigen Programmierertraum von sich automatisch generierenden Benutzerinterfaces recht nahe kommen: Nicht nur in der Hinsicht, dass das Interface vom Browser aus einer Zweckbeschreibung automatisch generiert wird. Aufgrund ihrer Deklarativität ist auch eine vollständig oder teilweise automatisierte Generie-

[22] Mit *X-Smiles* (`http://www.x-smiles.org/`) existiert bereits ein Browser, der einen wesentlichen Teil der XForms-Spezifikation implementiert.

[23] Voraussetzung dafür ist natürlich, dass auch die serverseitigen XML-Applikationen dazu in der Lage sind (vgl. Abschnitt 1.4).

rung der Zweckbeschreibungen selbst leicht zu realisieren. Da XForms auf XML basieren, können ohne größeren Aufwand XML-Werkzeuge, wie z. B. XSLT-Scripts, benutzt werden, um aus XML-Schemata-Dateien automatisch XForms-Beschreibungen zu erzeugen.

#### 1.6.3.2 Webevolution durch Redundanzvermeidung

Die »Last« der Abwicklung von konkreten Benutzerinteraktionen würde durch eine solch abstrakte Standardisierung von Webformularen dahin geschoben wo sie hingehört, nämlich zu den Webbrowsern. Der grundsätzliche Vorteil dieser Verschiebung ist, wie bei vielen neuen Webtechniken, dass die Anzahl der Kopien funktional ähnlicher Informationen auf das notwendige Maß reduziert wird, denn im Vergleich zu der Anzahl an Formularen im WWW ist die Anzahl verschiedener Browser sehr gering.

Die geringere Redundanz, bzw. die eingeschränktere Verteilung von Interaktionsmechanismen die bei Webformularen durch die Entkoppelung von Zweck und Form ermöglicht würde, erlaubt, ähnlich wie die Entkoppelung von Inhalt und Form, eine schnellere Evolution des WWW, da die Benutzbarkeit einer sehr großen Anzahl von Webformularen, allein durch die Weiterentwicklung einiger weniger Browser verbessert werden kann.

#### 1.6.3.3 XForms als universelles Interface für Webapplikationen

Die Lokalisierung der konkreten Interaktionsmechanismen im Browser sorgt darüber hinaus für eine stärkere Vereinheitlichung der Bedienung, aber gleichzeitig auch für deutlich mehr Möglichkeiten der benutzerspezifischen Individualisierung. Die Individualisierung der Interaktion kann dabei, wie Abbildung 1.2 zeigt, deutlich über die bisher übliche Auswahl von Themen, *GUI-Toolkits* oder eigenen CSS hinausgehen. Die abstrakte Ebene, auf der XForms Benutzerinterfaces beschreibt, schließt im Prinzip keine Art von Interaktionssituation aus. Virtuelle XForms-Maschinen sind sicherlich für Telefone, Mobiltelefone und Fernseher, aber auch für Interaktionssituationen ganz ohne menschliche Beteiligung denkbar. Durch die Typisierung der Daten, mögliche Festlegung von Datenschemata und die Möglichkeit der Strukturierung der Eingabedaten in XML-Dokumente konstituieren XForms ebenso wie *User*

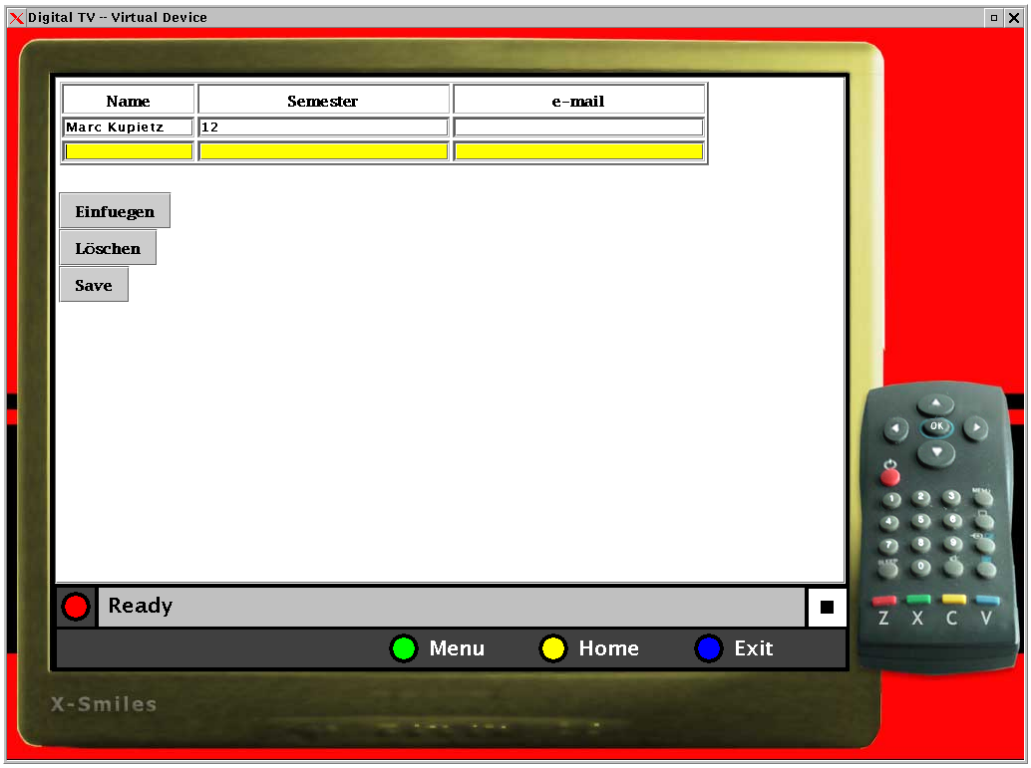


**Abbildung 1.2:** *X-Smiles DigiTV-Demo-Interface für XHTML und XForms.*

*Interfaces* (UIs) gleichzeitig auch *Application Programmer Interfaces* (APIs), die nicht nur das automatisierte Ausfüllen von Webformularen erleichtern, sondern zukünftig die Rolle des Standard-API von Webapplikationen einnehmen könnten. Das Anwendungsfeld für solche flexiblen APIs wäre weit: z. B. von der einfachen Übermittlung von Veranstaltungsdaten an Zeitschriften bis hin zu Geschäftsabwicklungen im Business-2-Business E-Commerce-Bereich.

# 2 Aufbereitung von Daten und Dokumenten

Nachdem im vorangegangenen Kapitel die grundlegenden Formalismen zur Auszeichnung, Repräsentation, Abfrage und Veränderung von Inhalten unterschiedlicher Art dargestellt wurden, sollen im Folgenden Programmiertechniken vorgestellt werden, mit Hilfe derer diese Inhalte für ihre Präsentation – insbesondere im *World Wide Web* – aufbereitet werden können.

Die grundsätzliche Aufgabe einer solchen *Aufbereitung* besteht darin, Daten unterschiedlicher Quellen und ihre Auszeichnungen – ob in Form von Datenbanken oder XML-Dokumenten repräsentiert – in eine für ein bestimmtes Zielmedium und einen intendierten Zweck adäquate Form zu transformieren. Das heißt z.B., dass die gemäß der Form-Inhalt-Trennung unabhängig von Gestaltungsinformationen vorliegenden Daten und Dokumente mit eben solchen Angaben zur ihrer Präsentation vereinigt werden.

## 2.1 Cascading Style Sheets

Eine erste Trennung von Inhalt und Form im WWW ermöglichte Ende 1996 die Verabschiedung des CSS-Standards (*Cascading Style Sheets*) durch das WWW-Konsortium [LB96]. In der ersten Version des Standards konnte der *Stil*, also z. B. Farbe, Größe, relative Positionierung, usw., von HTML-Elementen separat von den HTML-Dokumenten in sogenannten Stylesheets festgelegt werden. Das Problem bestand allerdings lange Zeit darin, dass zwar Inhalt und Form voneinander getrennt werden konnten, das Ergebnis der clientseitig durch den Browser vorgenommenen Vereinigung dieser beiden Informationsquellen zu einer Bildschirmdarstellung (*Rendering*) al-

lerdings nicht vorhersagbar war, da CSS von unterschiedlichen Browsern in sehr unterschiedlichem Maße und auf sehr unterschiedliche Art und Weise unterstützt wurde. Um möglichst browserunabhängige verlässliche *Rendering*-Resultate zu erzielen, wurden (und werden) daher stilistische Informationen sowohl in Stylesheets als auch zusätzlich – wie zuvor – in den HTML-Dokumenten selbst angegeben.

Mittlerweile hat sich die Situation dahingehend verbessert, dass sich zum einen mit der Verabschiedung des CSS2-Standards [CSS98] ihre Mächtigkeit vergrößert hat und zum anderen ein sehr großer Teil der aktuell verwendeten Browser in der Lage ist, CSS auch auf nicht HTML-konforme, allgemeine XML-Dokumente ausreichend spezifikationsgetreu anzuwenden. In Bezug auf die Aufbereitung von Inhalten haben *Cascading Stylesheets* jedoch zwei wichtige inhärente Einschränkungen:

1. CSS können die Struktur von Quelldokumenten nicht verändern. Das heißt, sie können lediglich Elemente auslassen, nicht aber ihre Reihenfolge verändern oder neue Elemente hinzufügen.

2. Sie bestimmen lediglich das Aussehen der Elemente *eines* Dokuments, können aber nicht Informationen aus mehreren Quellen zusammenfügen.

Da aufgrund dieser Einschränkungen die Struktur der Präsentation abhängig von der Struktur des Quelldokuments ist, kann mit Hilfe von *Cascading Style Sheets* allein keine vollständige Trennung von Inhalt und Präsentation für den Webbereich erzielt werden. Um z. B. flexibel Ansichten auf XML-Dokumente erzeugen zu können oder mehrere Datenquellen, z. B. Beschreibungen von Navigationsstrukturen, Texte und Datenbankinhalte in Präsentationen vereinigen zu können, sind also (zusätzlich) mächtigere Werkzeuge notwendig.

## 2.2 CGI-Programmierung

Eine häufig verwendete Technik zur dynamischen Erzeugung von Webseiten ist die Nutzung von Allzweck-Programmier- (z. B. C oder Java) und Script-

sprachen (z. B. Perl, TCL oder Python) über das *Common-Gateway-Interface* (CGI), das eine Schnittstelle zur Kommunikation zwischen Webserver und solchen Programmen und Scripts definiert. Aufgrund der Erzeugung von HTML durch den Programmcode und der dadurch bedingten starken Unterordnung von Inhalt und Layout unter die Programmierung eignen sich solche CGI-Programme insbesondere dann zur Generierung von Webseiten, wenn diese abhängig von veränderlichen Parametern sind. Typischerweise also dann, wenn Seitenelemente abhängig von Webformular-Eingaben auf Basis von Datenbanken generiert und in eine eher starre Layoutschablone eingefügt werden sollen.

Zur Verarbeitung von XML-kodierten Inhalten existieren für die meisten Script- und Programmiersprachen Bibliotheken, die Methoden zum Parsen sowie Methoden zur Abfrage, Navigation und Manipulation von XML-Bäumen und der darin enthaltenen Informationen zu Verfügung stellen. Die Schnittstellen zu diesen Bibliotheken sind dabei weitgehend standardisiert. Die beiden wichtigsten Standard-APIs sind zum einen die *Simple API for XML* (SAX) und zum anderen das *Document Object Model* (DOM) [DOM00].

Die SAX-API ist ereignisbasiert, das heißt, dass beim inkrementellen Parsen eines XML-Dokuments *Ereignisse*, wie z. B. »*Starttag des Elements* `eg` *gefunden*«, ausgelöst werden, die vom Verarbeitungsprogramm selektiv ausgewertet werden können. Der Ansatz der DOM-API hingegen ist nicht inkrementell, sondern *holistisch* und auf einer etwas höheren Abstraktionsebene angesiedelt: Der Parser, der z. B. auf einer SAX-Bibliothek basieren kann, erzeugt zunächst eine vollständige, dem Dokument entsprechende, baumartige Datenstruktur, die dann mit festgelegten Methoden abgefragt und manipuliert werden kann.

Dank dieser beiden Standards lassen sich also XML-Dokumente in vielen Programmiersprachen auf sehr ähnliche Art und Weise verarbeiten. Da DOM und SAX jedoch, wie bereits im Zusammenhang mit XML-Datenbanken diskutiert (in Abschnitt 1.5) wurde, auf einer sehr niedrigen Abstraktionsebene angesiedelt sind und die Programmierung entsprechend aufwendig ist, stellen Allzweck-Programmiersprachen zur *Aufbereitung* von XML-kodierten Inhalten für das Web, im Gegensatz zur *Generierung* von HTML-Seiten als Reaktion auf Benutzereingaben, keine optimale Lösung dar.

## 2.3 XSL - Transformations

Die 1999 als W3C-Empfehlung veröffentlichte *eXtensible Stylesheet Language: Transformations* [XSL99] wurde speziell zur Transformation von XML-Strukturen entwickelt und hat sich schnell als die Standardtechnik auf diesem Gebiet etabliert. Die grundsätzliche Funktionsweise einer XSL-Transformation ist folgende: ein *XSLT-Prozessor* nimmt als Eingabe jeweils ein XML-Dokument und ein sogenanntes *XSLT-Stylesheet* (ebenfalls ein wohlgeformtes XML-Dokument). Anhand der im Stylesheet angegebenen Regeln und Anweisungen wandelt der Prozessor das Eingabedokument in eine Ausgabe um. Die Struktur der Ausgabe – deren Format als HTML, XML oder Text spezifiziert werden kann – ist dabei von der Eingabestruktur weitgehend unabhängig. Das heißt die Transformation kann beliebige Einfügungen, Auslassungen, Reihenfolgeveränderungen sowie Veränderungen an Elementen umfassen. Bei der Transformation kann neben dem Eingabedokument auch auf andere XML-Dokumente zugegriffen werden; jedoch nur *lesend*.

Auf eine detaillierte Einführung in Syntax und Semantik von XSLT soll an dieser Stelle verzichtet werden. Es sei dazu auf die einschlägige Literatur [z.B. Kay00] verwiesen. Stattdessen sollen im Folgenden die Besonderheiten dargestellt werden, die XSLT für die Transformation von XML-Strukturen Standard-Programmiersprachen gegenüber überlegen machen.

### 2.3.1 Deklarative vs. imperative Programmierung

In herkömmlichen imperativen (oder prozeduralen) Programmiersprachen muss die *Sequenz* von Schritten, die vollzogen werden müssen, um ein Resultat zu produzieren, explizit spezifiziert werden. In deklarativen Sprachen dagegen beschreibt man lediglich die *Relation* von Variablen durch Funktionen und Inferenzregeln. Der Prozessor (ein Interpreter oder Compiler) einer deklarativen Sprache appliziert einen festgelegten Algorithmus auf diese Relationen, um das Ergebnis zu produzieren. Die bekanntesten Vertreter solcher deklarativer Sprachen sind logische Programmiersprachen wie *Prolog* oder funktionale Sprachen wie *Haskell*.

XSLT kann sowohl im Sinne einer herkömmlichen imperativen Sprache

*prozedural*, wie auch im Sinne einer deklarativen Sprache *regelbasiert* benutzt werden. Das heißt, XSLT stellt nicht nur Konstrukte zur Kontrolle des Programmflusses, wie z. B. Schleifen, if-then-Ausdrücke und Funktionsaufrufe zur Verfügung, sondern auch einen Mechanismus zur Regelverarbeitung.

Eine herkömmliche imperative Programmierung einer XML-Baum-Transformation bietet sich dann an, wenn die Struktur der Eingabe einfach und statisch ist. In diesem Fall können die Knoten, dank der bekannten Baumstruktur, explizit ausgewählt und der Reihe nach verarbeitet werden. Da sich XML-Sprachen aber gerade durch ihre flexible, rekursive Strukturierbarkeit und leichte Erweiterbarkeit z. B. gegenüber Datenbanken auszeichnen, ist der prozedurale Ansatz nur in wenigen Fällen eine adäquate Lösung. Gerade bei der Standard-Problemklasse, nämlich der Aufbereitung von XML-Dokumenten z. B. für das WWW, stoßen prozedurale Lösungsansätze schnell an ihre Grenzen. Selbst einfache Dokumenttypen erlauben durch verschachtelt definierte Elemente Bäume, deren Struktur nur schwer überschaubar bzw. vorhersagbar ist. Gleichzeitig wird der Programmcode, der notwendig ist um verschiedene Strukturen explizit zu klassifizieren und abzufangen, unüberschaubar und damit schwer zu modifizieren oder zu warten.

In deklarativen Programmen bzw. sogenannten regelbasierten XSLT-Stylesheets legt der Programmierer in erster Linie nur eine Menge von Produktionsregeln fest, von denen jede erstens ihre Anwendungsbedingungen und zweitens ihr Resultat spezifiziert, z.B.: »*wenn Du ein* `fliesstext`*-Element findest - mache daraus ein* `p`*-Element*«. Im Gegensatz zu prozeduralen Programmen, bzw. sogenannten *navigational Stylesheets* spiegelt die Anordnung und Struktur dieser Produktionsregeln dabei nicht notwendigerweise die Struktur des Eingabedokumentes wider. Tatsächlich sind die Annahmen, die bezüglich der Struktur der Eingabe gemacht werden müssen, minimal. Wenn die Sprache eines zu transformierenden Dokuments gegenüber der, für die ein deklaratives Stylesheet spezialisiert war, verändert oder erweitert wurde, ist deshalb die Wahrscheinlichkeit eines vollständigen Zusammenbruchs der Transformation generell geringer als bei einem entsprechenden *navigational Stylesheet*. Außerdem müssen bei regelbasierten Stylesheets, um Erweiterungen einer zu transformierenden XML-Sprache – z. B. bei lokalen Erweiterungen von Standard-DTDs – Rechnung zu tragen, typischerweise keine

bestehenden Code-Fragmente verändert werden, sondern entsprechend der Erweiterungen lediglich neue Regeln hinzugefügt werden. Diese Eigenschaft einer *additiven Semantik* spielt, insbesondere auch in Bezug auf die in Abschnitt 2.4 thematisierten Hierarchisierung von XSLT-Stylesheets, eine wichtige Rolle. Die Möglichkeit, Stylesheets regelbasiert formulieren zu können, trägt also dem erweiterbaren Charakter von XML-Sprachen besondere Rechnung.

Ein weiterer Vorteil der regelbasierten Programmierung in XSLT beruht darauf, dass der Mechanismus, der die Regeln verarbeitet, festgelegt ist. Ein Aspekt des Vorteils liegt auf der Hand: Es ist nicht notwendig, für jedes Problem einen neuen Verarbeitungsmechanismus zu programmieren. Der zweite Aspekt ist, dass der Mechanismus der Verarbeitung nur *einmal* verstanden werden muss und nicht für jedes existierende Programm aufs neue. Das heißt das regelbasierte XSLT-Stylesheets nicht nur kürzer und damit übersichtlicher sind, sondern in der Regel leichter verständlich und damit besser wartbar.

### 2.3.2 Funktionale Eigenschaften von XSLT

Eine Unterklasse der deklarativen Programmiersprachen sind, wie bereits erwähnt, die funktionalen Programmiersprachen. Ein Programm in einer funktionalen Sprache besteht aus einer Menge eventuell rekursiver Funktionsdefinitionen und eines Ausdrucks, dessen Wert dem Resultat des Programms entspricht. Der Mechanismus des Prozessors beruht dabei auf einem getypten Lambda-Kalkül mit Konstanten. Es gibt keine Nebeneffekte bei der Auswertung von Ausdrücken, so dass ein Ausdruck, z. B. eine Funktion angewendet auf bestimmte Argumente, immer den selben Wert ergibt (falls seine Auswertung terminiert). Außerdem kann ein Ausdruck an jeder Stelle durch seinen Wert ersetzt werden ohne das Gesamtresultat zu verändern (*Referentielle Transparenz*).

#### 2.3.2.1 Referentielle Transparenz

XSLT ist keine typische funktionale Programmiersprache, da sie Lambda-Ausdrücke bzw. höhergradige Funktionen (Funktionen mit Funktionen als Parameter) nicht direkt unterstützt.[1] XSLT erfüllt jedoch die wichtigste Eigenschaft einer funktionalen Sprache, nämlich referentielle Transparenz.

Eine notwendige Bedingung für die generelle Austauschbarkeit von Ausdrücken mit ihren Werten mag auf den ersten Blick als Nachteil erscheinen: In XSLT existieren keine Variablen im Sinne herkömmlicher Programmiersprachen. Wird der Wert einer Variable einmal definiert, so behält sie diesen. Eine erneute Zuweisung eines Wertes ist ein Fehler. Das heißt, XSLT-Variablen sind im streng mathematischen Sinne bzw. im Sinne von lokal gebundenen Konstanten zu verstehen.

Das Problem mit Variablenzuweisungen, wie sie in imperativen Programmiersprachen gebräuchlich sind, ist, dass sie einen sogenannten *Seiteneffekt* erzeugen, der der Ausführung des Programms bestimmte Restriktionen auferlegt. Wenn nämlich Funktionen, bzw. Prozeduren Seiteneffekte haben, also nicht nur einen Wert zurückliefern, sondern in irgendeiner Weise Veränderungen an der Umgebung vornehmen (z. B. eine Variable ändern, auf die andere Funktionen zugreifen können, etwas auf den Bildschirm ausgeben oder etwas in eine Datei schreiben), kommt es entscheidend auf die Reihenfolge und Häufigkeit an, mit der diese Funktionen ausgeführt werden. Seiteneffektfreie Funktionen, sogenannte *reine Funktionen*, dagegen können beliebig oft, in beliebiger Reihenfolge ausgeführt werden. Eine reine Funktion, die z. B. die Fläche eines Kreises berechnet, kann beliebig oft aufgerufen werden, ohne dass sich das Ergebnis ändert. Wenn sie dagegen einen Nebeneffekt hat, also z. B. den Radius des Kreises verändert, oder es auch nur nicht sicher ist, ob sie keinen Nebeneffekt hat, ist es entscheidend, wie oft und wann diese

[1] Dimitre Novatchev [Nov01] zeigt, dass Funktionen höherer Ordnung in XSLT über den Umweg sogenannter *Template-Referenz-Datentyps* implementierbar sind. Da XSLT beim Aufruf einer Funktion mit `xsl:call-template` keinen variablen Funktionsnamen zulässt, wird stattdessen `xsl: apply-templates` mit einem select-Attribut, das genau einen Knoten auswählt, der wiederum genau von dem Template gemacht wird, das die gewünschte Funktion berechnet. Novatchev zeigt, dass alle in [Hug00] aufgeführten, für funktionale Programmiersprachen charakteristischen Funktionen höherer Ordnung sich über diesen Umweg implementieren lassen und XSLT damit alle Eigenschaften einer funktionalen Programmiersprache besitzt.

Funktion aufgerufen wird.

XSLT-Templateregeln, die Hauptbestandteile von XSLT-Stylesheets, sind reine Funktionen, die die aktuelle Position im Eingabebaum und eventuell weitere Parameter auf eine Ausgabe abbilden. Die Reihenfolge, in der diese Funktionen definiert oder ausgewertet werden, spielt keine Rolle (solange der XSLT-Prozessor die Ergebnisse in der richtigen Reihenfolge wieder zusammensetzt). Aufgrund dieser Reihenfolge-Unabhängigkeit bei der Auswertung der Funktionen sind implizite, versteckte Annahmen über die Struktur von Eingabedokumenten in XSLT-Stylesheets ausgeschlossen. Fehler, die aus nicht »vorhergesehenen« Eingabestrukturen resultieren, können damit so weit wie möglich vermieden werden. Außerdem ist damit die Anpassung von XSLT-Stylesheets an neue Eingabedokument-Typen in der Regel deutlich einfacher als bei imperativen Transformationsprogrammen.

Neben diesen Vorteilen, die XSLT für den Programmierer zur ersten Wahl bei der Verarbeitung komplexer XML-Dateien machen, ergeben sich aus der Freiheit von Nebeneffekten Möglichkeiten zum *Caching* von Transformationsergebnissen (siehe dazu Abschnitt 4.4.2).

#### 2.3.2.2 Funktionale Geschlossenheit

Eine weitere interessante Eigenschaft von XSLT als Programmiersprache ist die der sogenannten *Geschlossenheit*[2]. Geschlossenheit bedeutet in diesem Zusammenhang, dass das Datenstruktur-Format der Eingabe dem der Ausgabe entspricht. Sowohl die Eingaben als auch die Ausgaben sind XML-Dokumente oder, abstrakt betrachtet, wohlgeformte XML-Bäume. Das hat den Vorteil, dass sich beliebig viele XML-Transformationen verketten lassen und damit komplexe Transformationen in einfachere Transformationsschritte aufspalten lassen.

Leider gilt beim aktuellen XSLT-Standard 1.0 die Eigenschaft der Geschlossenheit nur auf der Ebene kompletter Stylesheets. Das heißt, um eine Transformation in zwei Schritte zu zerlegen benötigt man zwei Stylesheets. Der Grund dafür ist, dass die meisten XPathünd XSLT-Funktionen bzw. Templa-

[2]Geschlossenheit ist keine generelle, sondern eine optionale Eigenschaft von XSLT-Stylesheets. Die Ausgabe eines wohlgeformten XML-Dokuments ist nur dann garantiert, wenn als Ausgabemethode `xml` und nicht `text` oder `html` gewählt wird.

tes als Eingabe-Datentyp einen sogenannten *node-set* verlangen, während ihre Ausgabe vom Typ *result-tree-fragment* ist. Wieso dies so ist, ist nicht nachzuvollziehen, da es sich beim *result-tree-fragment* um einen Baum handelt, der einem *node-set* mit einem Element, nämlich dem Wurzelknotens des Baums genau entspräche. Viele XSLT-Prozessoren stellen demzufolge auch eine Erweiterungsfunktion zur Verfügung, die ein *result-tree-fragment* in einen *node-set* umwandelt, so dass sich Templates innerhalb von Stylesheets verketten lassen. Im kommenden XSLT-Standard 2.0, dessen Spezifikation sich zur Zeit noch in der Entwurfsphase befindet, wird das *result-tree-fragment* sehr wahrscheinlich durch den Typ *node-set* ersetzt sein [siehe XSL01b].

XSLT erfüllt nebenbei das Kriterium der Geschlossenheit noch in einem weiteren Sinne. Nicht nur bei den Ein- und Ausgaben handelt es sich um XML-Dokumente, sondern auch bei den Stylesheets selbst. So lassen sich z. B. sehr leicht Dokumentationen von Stylesheets aus den Stylesheets selbst wiederum mit Hilfe von Stylesheets generieren.[3] Als noch interessantere Anwendungsmöglichkeit dieses Eigenschaft wäre es denkbar, mit Hilfe von XSLT-Stylesheets wiederum XSLT-Stylesheets zu erzeugen.

## 2.4 Modularisierung, Hierarchisierung und Vererbung

*Vererbung* ist ein Konzept, das hauptsächlich von objektorientierten Programmiersprachen, wie z. B. JAVA oder C++, bekannt ist. Diese Sprachen ermöglichen es dem Programmierer, verschiedene Klassen zueinander in hierarchische Beziehungen zu setzen, bzw. in Hierarchien anzuordnen, so dass untergeordnete Klassen (sogenannte *Spezialisierungen*) Eigenschaften ihrer übergeordneten Klasse (oder Klassen) ererben oder überschreiben.

XSLT ist keine objektorientierte Programmiersprache und verfügt auch nicht über ein Vererbungskonzept von Klasseneigenschaften, was aufgrund des eng umrissenen Aufgabenfelds für XSLT-Stylesheets auch nicht notwen-

[3] Ein Beispiel zur Generierung von Dokumentationen von XSLT-Stylesheets durch XSLT-Stylesheets findet sich in Abschnitt 5.5.4.

dig wäre. Ein Problem, das aber häufig bei der Aufbereitung von Daten auftaucht, ist, dass von Fall zu Fall Präsentationen erwünscht sind, die sich im Wesentlichen gleichen, aber sich auch in einigen Punkten unterscheiden. Dieses Dokument wurde zum Beispiel auf Grundlage eines für Benutzerhandbücher vorgesehen Stylesheets in seine Print"bzw. Webdarstellung transformiert. Ein großer Teil des Stylesheets konnte dabei übernommen werden, während einige Eigenschaften (z. B. Titelformatierung, Zitierweise, Darstellung von Programm-Code) verändert oder erweitert werden mussten. Bei größeren Webauftritten wir das Problem noch deutlicher: Ausgehend von einem einheitlichen Corporate Design eines Konzerns oder einer öffentlichen Einrichtung werden Sub-Identitäten mit charakteristischen Eigenschaften für Sub-Unternehmen bzw. für Sub-Einrichtungen benötigt (siehe auch Abschnitt 5.5.2).

Natürlich wäre es möglich, für jede Sub- und Subsub-Identität modifizierte Kopien der XSLT-Programme zu erzeugen. Dies würde allerdings zu einer ähnlichen Informationsredundanz führen, die auf niedrigerer Ebene ja gerade durch die Trennung von Inhalt und Form und durch die Verwendung von Stylesheets vermieden wird (siehe dazu Abschnitt 1.1.4). Jede Änderung und Erweiterung müsste wieder an jeder Kopie vorgenommen werden. Konsistenz wäre nicht gewährleistet. Auch eine einfache Parametrisierung veränderlicher Eigenschaften scheidet als Lösungskanditat aus, da kaum vorhersagbar ist, was alles im Laufe der Zeit verändert wird.

Die einzige Lösung, die dem Problem gerecht wird, ist eine Form von Vererbungshierarchie. XSLT bietet diese Möglichkeit zwar nicht direkt an, Befehle zur Festlegung der Präzedenz von Regeln, Funktionen und Variableninitialisierung ermöglichen jedoch eine zur klassischen Vererbungskonzepten (fast) äquivalente Funktionalität zu erzielen.

Der in diesem Zusammenhang wichtigste XSL-Befehl ist `xsl:import`. Dieser importiert ein XSLT-Stylesheet und weist den importierten Regeln, Funktionen und Variableninitialisierungen eine geringere Präzedenz zu als denen des aufrufenden Stylesheets. Auf diese Art und Weise lässt sich ein zum Ererben und Überschreiben geerbter Methoden und Konstanten äquivalenter Effekt erzielen. Da importierte Dateien wiederum `xsl:import`-Befehle enthalten können, lässt sich auch eine Form von Vererbungshierarchie aufbauen.

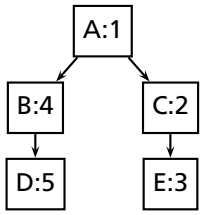


**Abbildung 2.1:** *Importpräzedenz bei XSLT.*
Stylesheet A importiert B und C. B importiert D, C importiert E. Die resultierende Präzedenzreihenfolge nach der alternative Regeln der Stylesheets ausgewählt werden ist: A, C, E, B, D.

Im Gegensatz z. B. zu JAVA ist dabei auch eine multiple Vererbung möglich: Stylesheets können – mit zunehmender Präzedenz – mehrere Stylesheets importieren (siehe Abbildung 2.1).

XSLT bietet ähnlich wie objektorientierte Sprachen auch die Möglichkeit, überschriebene Funktionen einer Oberklasse anzuwenden. Die Funktion `xsl:apply-imports` verhält sich in etwa analog zur `super`-Funktion in Java. Im Gegensatz zu Java erlaubt der aktuelle XSLT-Standard jedoch nicht, importierte Regeln mit Parametern aufzurufen. Falls eine Regel mit Parametern aufgerufen wird, so nehmen diese ihre Defaultwerte an. Diese extreme Einschränkung in der Erweiterung von Oberklassen wird zwar wahrscheinlich in XSLT 2.0 aufgehoben, jedoch wird es auch in dem kommenden Standard weiterhin weder möglich sein, importierte benannte Regeln (also Funktionen) aufzurufen, noch den »aktuellen Knoten« für den Aufruf einer importierten Regel per `select` auszuwählen, noch den Modus des Aufrufs per `mode`-Attribut auszuwählen. Letztendlich kann ein zum Aufruf überschriebener Funktionen äquivalentes Verhalten nur über einen Umweg in etwa folgendermaßen realisiert werden: Funktionen, die ihre überschriebene Pendants aufrufen sollen, müssen nicht als benannte Templates, sondern als Modus-sensitive Templates implementiert werden. Etwaige Parameter müssen (zumindest im zur Zeit maßgeblichen XSLT 1.0) in Form eines *`node-set`* per select übergeben sein und können damit für den Aufruf überschriebener

Templates nicht verändert werden.[4]

## 2.5 XSL - Formatting Objects

Ebenso wie XSLT sind *XSL – Formatting Objects* (XSL-FO) [XSL01a] Teil der *eXtensible Style Sheet Language* XSL. Während XSLT, wie zuvor dargestellt, generelle XML-Transformationen beschreibt, dient XSL-FO zur *medienspezifischen Beschreibung des Erscheinungsbilds* von Dokumenten. Die Ebene der Beschreibung liegt dabei über der von Seitenbeschreibungssprachen wie Postscript. So können unterschiedliche Layoutregeln spezifiziert werden – z. B. wo Seitenumbrüche zulässig sind – nicht jedoch die genaue Platzierung von Elementen auf einer Seite. Die Berechnung solcher Angaben ist die Aufgabe sogenannter *Formatter*, die XSL-FO z. B. in Seitenbeschreibungssprachen wie Postscript oder PDF übersetzen.

Die XSL-FO-Formatierungs-Anweisungen und Parameter sind namentlich so weit wie möglich an die der *Cascading Style Sheets* angelehnt. Die Möglichkeiten von XSL-FO sind jedoch deutlich umfangreicher. So entspricht die Struktur eines XSL-FO-Ausgangsdokuments nicht notwendigerweise der des Zieldokuments (siehe Abschnitt 2.1). Darüber hinaus kann mit XSL-FO z. B. Inhalt und Aussehen kompletter Büchern festgelegt werden, während CSS eher dazu geeignet sind, Stil und Layout von Elementen von HTML-Seiten festzulegen.

Der typische XSL-Formatierungs-Objekte involvierende Produktionsprozess ist folgender: Anhand von XSLT-Stylesheets transformiert ein XSLT-Prozessor ein XML-Dokument in eine XSL-FO-Ausgabe. Diese wird dann von einem *Formatter* in zur Ansicht oder zum Druck bestimmte Formate umgewandelt. Das heißt, dass auf Basis eines generisch ausgezeichneten XML-Dokuments – geeignete XSLT-Stylesheets vorausgesetzt – automatisch sowohl für das WWW als auch für den Buch- oder Zeitschriftendruck optimierte Ausgaben erzeugt werden können und damit sogenanntes *Single Source (Cross Media) Publishing* ermöglicht wird.

[4] siehe Abschnitt 5.5.2 für einen Anwendungsfall

## 2.6 HTML-eingebettete Scriptsprachen

Eine weitere Klasse von häufig zur Aufbereitung von Webseiten eingesetzten Scriptsprachen sind solche, die direkt in den Quelltext von HTML-Seiten eingebettet und vom Webserver vor der Auslieferung einer Seite verarbeitet werden. Der älteste Vertreter solcher HTML-Einbettungen sind sogenannte *Server Site Includes*, die in verschiedenen Varianten von den meisten gebräuchlichen Webservern unterstützt werden. Inzwischen haben sich aber eine Reihe weiterer auf diesem Prinzip basierender, z.T. sehr mächtiger Programmiersprachen (z. B. PHP, DTML, CFML) sowie Schnittstellen zu existierenden Allzweck-Programmiersprachen (z. B. JSP oder Java SSI) entwickelt.

Serverseitig interpretierte HTML-Einbettungen eignen sich vor allem um einzelne Elemente in Webseiten zu integrieren, die erst zum Zeitpunkt ihrer Auslieferung an einen Client berechnet werden können oder sollen. Bei solchen Elementen kann es sich z. B. um Zugriffszähler, um die Ergebnisse von Datenbankabfragen oder um Elemente handeln, die von in der HTTP-Anfrage übergebenen Parametern abhängig sind. Ein Vorteil gegenüber CGI-Scripts ist vor allem dann gegeben, wenn die auszuliefernden Seiten nur zu einem geringen Teil Ergebnis dynamischer Berechnungen sind, da der Quelltext den Charakter eines HTML-Dokuments behält und er als solcher in der Regel weiterhin mit HTML-Editoren bearbeitet werden kann.

Diese enge Verknüpfung von Programmlogik, HTML-Layout-Beschreibung und Inhalten innerhalb eines einzelnen Dokuments scheint auf den ersten Blick den Prinzipien einer möglichst weitgehenden Trennung von Inhalt und Präsentation zu widersprechen. Eine solche Trennung kann jedoch erreicht werden, indem die um Scripts erweiterten HTML-Dokumente als Schablonen (*Templates*) verwendet werden, in die die eigentlichen Inhalte (unter Umständen als Ergebnis dynamischer Berechnung) ebenso wie andere dynamische Elemente zum Zeitpunkt der Abfrage einer Seite eingefügt werden. Während dieses Verfahren die Aufgabe einer Präsentations-/ Inhalts-Entkoppelung zu einem gewissen Grad lösen kann, ist es allerdings – aus den im Zusammenhang mit CGI-Scripts und XSLT oben genannten Gründen – als *alleinige* Lösung zur Transformation bzw. Aufbereitung generisch annotierter XML-Dokumente wenig geeignet.

## 2.7 Fazit

Wie in Abschnitt 2.3 ausführlich dargelegt wurde ist XSLT als Werkzeug zur Übersetzung generischer Auszeichnungssprachen in darstellungsorientierte Auszeichnungssprachen, wie HTML oder XSL-FO, konkurrenzlos. Da im Web darzustellende *Inhalte* jedoch typischerweise nicht ausschließlich als wohlgeformte XML-Dokumente vorliegen, sondern z. B. auch als Datenbankeinträge oder Parameter von HTTP-Anfragen, reicht XSLT als alleiniger Aufbereitungsmechanismus nicht aus.

Um sowohl solchen Daten als auch Dokumenten gerecht zu werden, bietet sich als generelle Lösung des Aufbereitungsproblem daher eine, wie in Abbildung 2.2 dargestellte, mehrstufige Architektur an: In der ersten Stufe werden XML-Dokumente zunächst über XSLT-Stylesheets entweder in HTML-Dokumente oder in XSL-FO-Dokumente mit ggf. eingebetteten Scripts transformiert. In der zweiten Stufe werden die eingebetteten Scripts von einem Scriptprozessor interpretiert und ausgeführt. Resultierende *reine* HTML-Dokumente werden dann direkt an den Browser weitergeleitet. Resultierende XSL-FO-Dokumente werden zuvor noch in einem dritten Transformationsschritt durch einen *FO-Formatter* nach PDF umgewandelt.

Auf Grundlage einer solchen Architektur können sowohl dokumentorientierte als auch datenorientierte oder »dynamische« Inhalte mit den jeweils geeigneten Methoden sowohl für die Browserdarstellung als auch für die Druckausgabe integriert und aufbereitet werden.

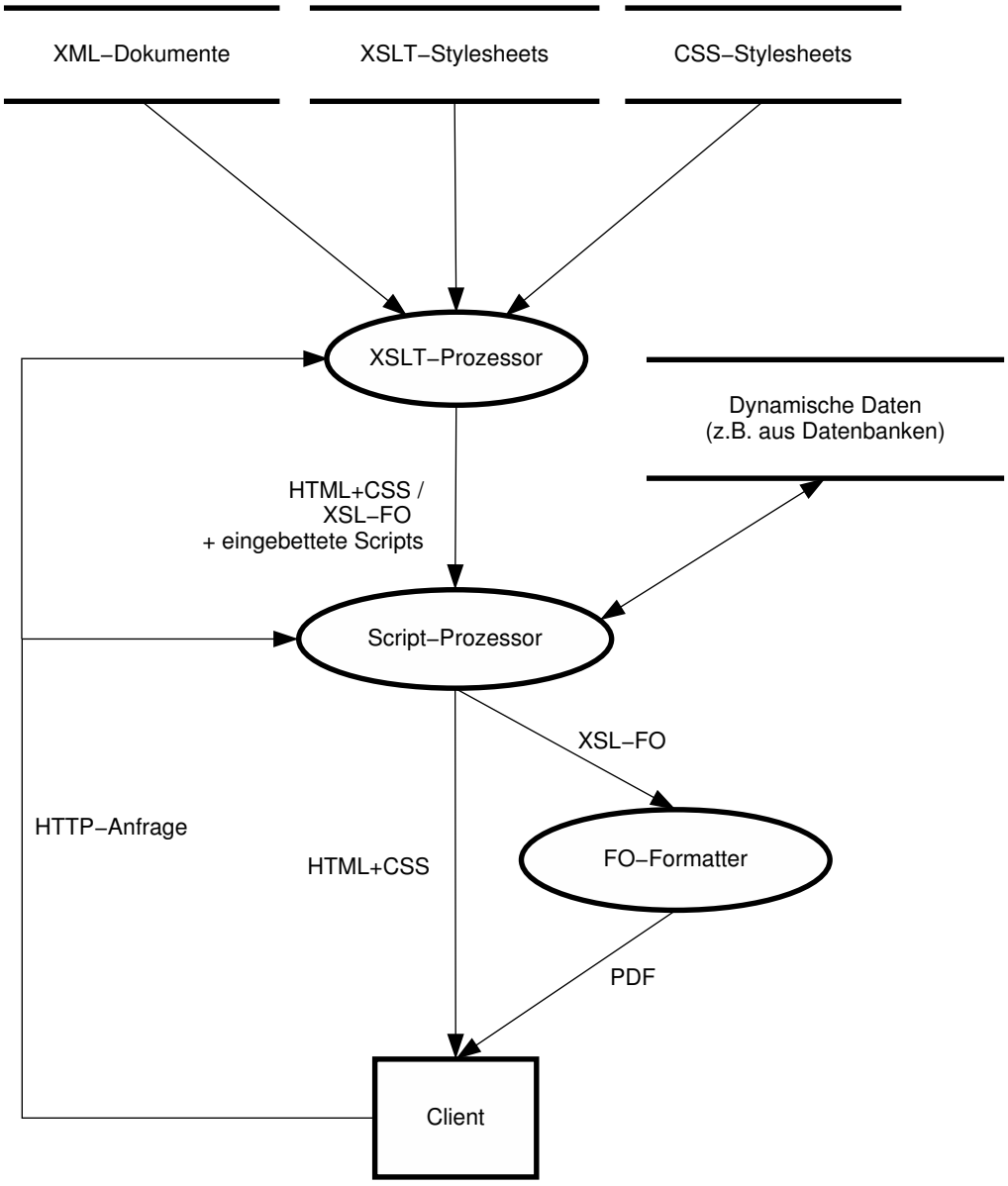


**Abbildung 2.2:** Datenfluss-Architektur zur Integration und Aufbereitung von Dokument- und Daten-orientierten Inhalten

# 3 Semantische Erweiterungen des World Wide Web

Während in den vorangegangenen zwei Kapiteln die Repräsentation von Informationen und deren Aufbereitung für den *menschlichen* Betrachter thematisiert waren, soll es in diesem Kapitel darum gehen, wie Informationen so repräsentiert oder aufbereitet werden können, dass sie *»verständlich«* für *Maschinen* sind. *Verständlich* soll heißen, dass eine automatische Verarbeitung nicht allein anhand *syntaktischer* Eigenschaften, sondern auch aufgrund von *semantischen* Beziehungen ermöglicht werden soll. Das vom W3-Konsortium mit *Semantic Web* titulierte Ziel, ist dabei ein globales Informationsnetz, in dem eine solche automatisierte, semantische Weiterverarbeitbarkeit über spezielle Anwendungen oder Domänen hinweg möglich wird.

## 3.1 Semantische Interoperabilität

Der entscheidende Faktor für den Erfolg fast aller Internetanwendungen ist ihre Verbreitung oder andersherum gesagt:

> *»It's not cool to be different, at least not when Internet computing is concerned.«* Edd Dumbill [Dum00]

Während bei einigen Internetformaten und Protokollen weitgehende Einigkeit besteht, z. B. beim *Hyper Text Transfer Protocol (HTTP)* und bei der *Hyper Text Markup Language* (HTML) bleiben andere Dienste und Datenressourcen scheinbar hoffnungslos unverbunden. Will man z. B. die Ergebnisse einer Filmrecherche in der *Internet Movie Database* über das Webinterface einer lokalen Videothek mit den aktuell ausleihbaren Filmen abgleichen, so

bleibt einem normalerweise keine andere Möglichkeit, als die Titel aus der Ergebnisliste per *cut and paste* in die Eingabemaske zu übertragen und jeden einzelnen Film »von Hand« zu suchen. Gleiches gilt z. B. für einen Studierenden, der sich mit Hilfe des elektronischen Vorlesungsverzeichnisses seiner Universität einen Semesterplan zusammengestellt hat und die empfohlene Semesterliteratur mit dem aktuell ausleihbaren Bestand seiner Universitäts-Bibliothek abgleichen will. Es sind also fast immer manuelle Zwischenschritte notwendig, wenn man die Funktionen mehrerer internetbasierter Applikationen miteinander Verknüpfen möchte.

Dasselbe gilt meist auch für die Verkettung von Internetapplikation und lokalen Anwendungen: Will derselbe Student die Daten seiner Veranstaltungen in seinen Terminkalender übernehmen, so muss er sie in der Regel auch von Hand übertragen.

Selbst zwischen lokalen Anwendungen ist die Situation nicht viel besser. Die häufig proprietären Datenformate lassen sich meist nur mit Programmen des selben Herstellers ohne wesentlichen Informationsverlust in einander überführen. Häufig führt der mit dem Export oder der Konvertierung einhergehende Informationsverlust zu einer zunehmenden Einschränkung der Verwendungsmöglichkeiten der Daten, so dass diese irgendwann in einer »Falle« gefangen sind und sich z. B. nur noch mit einem PDF- oder HTML-Browser betrachten oder ausdrucken lassen und nicht ohne weiteres mehr in ein editierbares Format überführbar sind.

Allgemein lässt sich das Problem so zusammenfassen, dass sich im Internet und – in Extension des Internets – lokal vorhandene Inhalte nur zu einem Bruchteil ihres Potentials automatisiert nutzen lassen. Neben dem Umstand, dass eine allgemeine Verfügbarkeit und automatische Verarbeitbarkeit von Inhalten natürlich manchmal z. B. aus wirtschaftlichen Interessen oder Datenschutzgründen nicht erwünscht ist, ist der Grund für diesen wenig zufriedenstellenden Zustand in den zur Zeit verwendeten Daten- bzw. Protokollformaten zu suchen, in den Inhalte abgelegt bzw. mit deren Hilfe sie verfügbar gemacht werden.

Das *World Wide Web* war ursprünglich als Raum für den Informationsaustausch zwischen Menschen gedacht. HTML war die Sprache, die in Verbindung mit dem Transferprotokoll HTTP den Anforderungen zunächst in aus-

reichendem Maße gerecht wurde. Auf der einen Seite war sie einfach genug um schnell Verbreitung zu finden, auf der anderen Seite mächtig genug, um eine grafische Darstellung von Inhalten ausreichend präzise und flexibel zu spezifizieren.

HTML besaß auch von Beginn an die Möglichkeit, Inhalte nicht nur grafisch, sondern auch generisch zu annotieren und mit zusätzlichen *Metainformationen* zu versehen. Diese Eigenschaften traten allerdings mit dem Einzug und der zunehmenden Verbreitung von WYSIWYG-HTML-Editoren immer weiter in den Hintergrund. Diese Entwicklung war aber auch durchaus folgerichtig, da das WWW praktisch ausschließlich zur direkten Mensch-Mensch-Kommunikation genutzt wurde und demnach hauptsächlich die für diese Art der Kommunikation geeigneten Kanäle, nämlich im Wesentlichen der visuelle relevant war. Wenn also zwischen der Browserdarstellung von:

```
<h1>Überschrift</h1>
```

und

```
<br/><b><font size="18">Überschrift</font></b><br/>
```

kein relevanter visueller Unterschied bestand, waren ausgehend von der obigen Prämisse beide Ausdrücke bedeutungsgleich und gegeneinander austauschbar.

Das Problem mit einer rein grafisch orientierten Auszeichnung ist für die Benutzung im Web (ähnlich wie in Abschnitt 1.1.4 aus Autorensicht beschrieben), dass damit die Verwendungsart auf die Anzeige im Browser eingeschränkt wird. Eine automatische Weiterverarbeitung ist so gut wie ausgeschlossen. Die graphischen Informationen selbst sind für Computerprogramme unverständlich, die Rekonstruktion der potentiell zugrundeliegenden strukturellen oder inhaltlichen Informationen ist nicht möglich, da vom grafisch orientierten Markup zu diesen keine eindeutige Relation existiert.

Das soll allerdings nicht heißen, dass mit einer generischen Verwendung von HTML-Tags der universellen Weiterverarbeitung von Webangeboten plötzlich Tür und Tor geöffnet wären. Das HTML-Vokabular, insbesondere das generische, ist zu klein, um Dokumente ausreichend spezifisch und

flexibel mit Zusatzinformationen zu annotieren. Außerdem steht seine Abgeschlossenheit an sich einer domänenunspezifischen, universellen und erweiterbaren Annotation im Weg.

XML scheint genau dieses Problem lösen zu können. Das Vokabular von XML-Sprachen ist beliebig erweiterbar, während die Syntax von XML-annotierten Dokumenten anhand von DTDs und XML-Schema auf ihre Korrektheit überprüfbar ist. Mittels spezifischer XML-Dokumententypen lassen sich so Inhalte exakt beschreiben und sicher zwischen Anwendungen austauschen.

Während XML als Formalismus zur Annotation von Ausgangsdaten und als Datenaustauschformat, wie in Abschnitt 1.1 beschrieben, viele Probleme löst, ist es auf dem Weg zu einem homogenen Web mit universell verwendbaren Inhalten nur ein erster Schritt. Der XML-Standard legt auf der Metaebene den Aufbau von XML-Dokumenten und DTDs exakt fest und beschreibt, wie die Syntax von XML-Markup-Sprachen mit Hilfe von DTDs definiert werden kann. Er beschreibt jedoch die *Bedeutung* von XML-Dokumenten nicht und stellt kaum Möglichkeiten bereit, die *Semantik* von XML-Sprachen formal zu spezifizieren. Was die einzelnen Elemente, Attribute und Texte und ihre Positionen innerhalb eines Dokuments bedeuten oder wie sie zu interpretieren sind, kann nicht innerhalb des XML-Formalismus selbst ausgedrückt werden. Die Semantik von XML-Sprachen kann nur mit Hilfe externer Formalismen oder informeller Beschreibungen festgelegt werden.

Bei der Verwendung von XML als Datenaustauschformat sind es die Programme oberhalb der Parserebene, die durch die Art ihrer Verarbeitung eine Prozess-Semantik konstituieren. Im Webbereich sind es in der Regel entweder XSLT-Scripts, die XML-Dokumente auf die hauptsächlich grafische HTML-Semantik zurückführen oder CSS-Stylesheets die mehr oder weniger direkt den einzelnen Elementen oder Attributen eine bestimmte grafische Bedeutung zuordnen. In beiden Fällen ist für die Entwicklung des Web hin zu einem homogenen Pool universell und *automatisch* weiterverarbeitbarer Informationen wenig gewonnen. Im Falle der Zurückführbarkeit auf eine grafische Semantik wird gegenüber einfachem HTML kein Vorteil erzielt. In allen anderen Fällen setzt die netzweite Verwertbarkeit der Informationen eine weite Verbreitung von Applikationen voraus, die die zugrundeliegende

XML-Sprache verarbeiten können. Zwar existiert eine große Anzahl von *standardisierten* XML-Sprachen bei denen die Bedeutung von Elementen oder Attributen beschrieben ist, die Beschreibung der Bedeutung ist aber nur informell und von der Interpretation durch einen menschlichen Leser abhängig. Formal besteht zunächst keine Möglichkeit, Bedeutungen oder Bedeutungsrelationen, wie z. B. dass ein Element Namens `h1` zur Klasse der *Überschriften* gehört, festzulegen. Die Konsistenz der informell beschriebenen Semantik solcher Sprachen mit den verschiedenen Semantiken, die durch auf sie spezialisierte Applikationen konstituiert werden, lässt sich also nicht kontrollieren. Aufgrund der fehlenden Möglichkeit, Bedeutungsrelationen festzulegen, führen außerdem kleine Änderungen oder Erweiterungen im Vokabular von XML-Sprachen bei ihrer Interpretation durch auf sie spezialisierte Applikationen nicht zu einer sogenannten *graceful degradation*, sondern zu einem partiellen oder vollständigen Zusammenbruch. *Semantische Interoperabilität* zwischen Anwendungen von XML-Sprachen ist also weniger eine graduelle, sondern eher eine *boolsche* Eigenschaft. Daraus folgt, dass XML-Sprachen entweder nur sehr langsam und zentral – also ohne von einem Netzwerkeffekt zu profitieren – weiterentwickelt werden können oder ihre automatisierte Verwertbarkeit geopfert wird. Eine semantische Interoperabilität lässt sich also selbst innerhalb eng umgrenzter Domänen nur schwer bzw. nur unter hohen Kosten aufrecht erhalten. Eine gewünschte domänenübergreifende und domänenvernetzende automatisierte Nutzung von Informationen ist innerhalb eines reinen XML-Frameworks nicht möglich.

### 3.1.1 Semantik von Auszeichnungssprachen – ein historischer Exkurs

XML-Dokumente vermitteln auf den ersten Blick einen Eindruck semantischer Transparenz. Wenn man eine XML-Datei in einem Editor öffnet und sie zum Beispiel mit Word- Postscript- oder Dateien in anderen proprietären Formaten vergleicht, so scheint die XML-Datei in der Regel mehr Sinn zu ergeben. Markup-Tag-Paare sagen uns welche Bedeutung eingeschlossene Text- oder Datenobjekte haben und wie sich diese Objekte zu größeren zusammenhängenden Einheiten strukturieren. So angenehm diese deskriptive Art von

Markup für den menschlichen Leser ist, für die maschinelle Verarbeitung, oder genauer gesagt für den maschinellen Informationsaustausch ist sie nur von begrenzter Relevanz.

> »*As enchanting as it is to contemplate the apparent ›semantic‹ clarity, flexibility, and extensibility of XML* vis-à-vis *HTML (e.g., how wonderfully perspicuous XML <bookTitle> seems when compared to HTML <i>), we must reckon with the cold fact that XML does not of itself enable blind interchange or information reuse. XML may help humans predict what information might lie ›between the tags‹ in the case of <trunk>, but XML can only help. For an XML processor, <trunk> and <i> and <bookTitle> are all equally (and totally) meaningless. Yes,* meaningless.« Robin Cover [Cov98]

XML Prozessoren haben in der Tat keine Möglichkeit, die inhärente Objektsemantik von XML-Dateien zu »verstehen«, da XML-Markup-Sprachen über keine vordefinierte Semantik verfügen. Die Typen von Relationen die sich transparent (d. h. eins-zu-eins) in XML ausdrücken lassen sind:

- *Enthalten sein* (A enthält B)
- *Hierarchie*
- *Adjazenz* (A folgt B)
- *Kookkurenz* (wenn A, dann [auch/nicht] B)
- *Attribut* (sehr eingeschränkt, da Attribute nur einen Wert vom Typ ID oder Zeichenkette haben können)
- *Referenz*

Während sich diese Konstrukte meist gut für Serialisierung von Objekten eignen, eignen sie sich kaum zur Modellierung von Objekten und deren Relationen innerhalb einer Domäne. Und selbst wenn man es schafft, semantische Relationen auf diese sehr einfachen syntaktischen Strukturen abzubilden, wäre ein XML-Prozessor ohnehin nicht in der Lage, ihre Bedeutung zu erkennen.

Die »*Blindheit auf dem semantischen Auge*« lag bei der Entwicklung der XML-Mutter SGML darin begründet, dass das primäre Ziel darin bestand, eine universelle und möglichst einfache Meta-Markup-Sprache zu schaffen und weniger eine Meta-Sprache, in der sich bestimmte Relationen möglichst einfach darstellen lassen. Außerdem wurde SGML zunächst nur zur Annotation von für den Druck vorgesehenen Texten verwendet. Das heißt, SGML wurde hauptsächlich in einer Domäne mit nur einem relevanten Datentyp genutzt, der zudem einen äußerst sequentiellen Charakter besitzt. Dass SGML oder ein Nachfahre von SGML jemals zu einem universellen Repräsentationsformalismus in einem »*World Wide Web*« werden sollte, konnte Anfang der 80er Jahre sicherlich niemand ahnen. Andererseits erwiesen sich bestimmte Designeigenschaften SGMLs in angrenzenden Domänen, bzw. insbesondere dann, wenn die annotierten Texte nicht mehr ausschließlich für den Druck verwendet werden sollten, schnell als hinderlich, unzureichend oder *unpassend*. So erwiesen sich die Möglichkeiten zur Festlegung semantischer Constraints zur Gewährleistung der Integrität z. B. literatur- oder sprachwissenschaftlich annotierter Texte in der Funktion von Forschungsdatenbasen als unzureichend; die in Hinblick auf das Ziel »Druck« klare Entscheidung zwischen Element (wird gedruckt) und Attribut (wird nicht gedruckt) als unpassend und die hierarchisch sequentielle Struktur für die Repräsentation »realer« Phänomene wie z. B. verbale Kommunikation als unzureichend und hinderlich.

Einen weiteren Grund für die »Aversion« SGMLs gegen Semantik liegt laut Robin Cover [Cov00] darin, dass die SGML-Bewegung ihre Opposition gegen Markup mit prozeduraler oder präsentationeller Semantik (ungerechtfertigterweise) gegen Semantik schlechthin verallgemeinert hat. Ausgehen von dem absolut berechtigten Credo:

> »*...the specification for the logical structure of the document should be made independent of and uncontaminated by any specification for the processing of that structure and content. And so, descriptive markup should be innocent of processing semantics.*«

wurde »*processing semantics*« langsam zu »*semantics*« verkürzt. Aussagen wie:

> »*Pure SGML languages do not countenance (processing) semantics... they should not try to express semantics... so design your processing semantics elsewhere...*«

täuschten über die Tatsache hinweg, dass zwischen der Deklaration, *was (prozedural) getan werden soll*, und der deklarativen Formulierung einer primitiven Objektontologie- und Objektrelationssemantik ein großer Unterschied besteht und dass der berechtigte Verzicht auf Ersteres Letzteres nicht zwangsläufig ausschließt.

In Bezug auf *Datentypsemantik* hat der Einzug von XML-Schema (siehe Abschnitt 1.1.1) inzwischen bewiesen, dass eine einfache (ontologische, relationale) Semantik sich durchaus innerhalb eines deklarativen Systems beschreiben lässt. Jedoch werden auch die hierarchischen Datentyprelationen in XML-Schema im Wesentlichen in Bezug auf ihre syntaktischen Implikationen ausgenutzt.

## 3.2 Auffindbarkeit von Informationen

Neben den oben genannten Forderungen einer semantischen Interoperabilität von Datenformaten ist eine verbesserte Auffindbarkeit von Informationen die zweite wichtige Voraussetzung für das Web der nächsten Generation.

### 3.2.1 Suchmaschinen und Kataloge

Während Volltext-Suchmaschinen, wie z. B. Google[1], in der direkten Mensch-Applikations-Kommunikation in vielen Fällen mit etwas Aufwand zu den gewünschten Informationen führen, sind sie für ein automatisches Auffinden von Informationen, also in Szenarien in denen Webdienste und Webapplikationen ohne menschliche Intervention verkettet werden sollen, unbrauchbar. Um bestimmte Dokumente mit Hilfe von Volltext-Suchmaschinen im Netz zu finden, müssen die Suchbegriffe anhand von Weltwissen über die in solchen Dokumenten wahrscheinlich enthaltenen Wörter und Phrasen, angepasst an eine angenommene Varianz und Häufigkeit, ausgewählt werden.

[1] `http://www.google.de`

Dann muss abhängig von der Anzahl und Qualität der Treffer entweder die Suche mit angepassten Suchbegriffen wiederholt werden oder die Trefferliste »zu Fuß« durchgegangen werden, um die unbrauchbaren Links auszusortieren. Sind bestimmte Informationen, z. B. die Daten aller Publikationen eines bestimmten Autors, gesucht, müssen die ausgewählten Dokumente noch weiter durchsucht werden. Um diese Vorgehensweise zu automatisieren, wären zum einen KI-Techniken, wie z. B. Planen, automatisches Sprachverstehen und eine große Wissensbasis nötig. Doch selbst mit großem Aufwand könnten Applikationen wohl zur Zeit nur innerhalb einer sehr eng umrissenen Domäne selbstständig im Web nach Informationen suchen.

Die alternative Methode, um Dinge im Netz zu finden, ist die Benutzung von Webkatalogen, wie z. B. Yahoo[2]. Dieser Weg bietet sich insbesondere dann an, wenn das Gesuchte nicht genau umrissen werden kann. Unumgänglich ist er, wenn sich keine Menge von Suchbegriffen für eine Anfrage an Volltext-Suchmaschinen bilden lässt, z. B. weil die gesuchten Dokumente keine charakteristischen Begriffe beinhalten. Für den menschlichen Benutzer haben solche Kataloge zunächst den Nachteil, dass er sich in jedem Fall durch mehrere Hierarchieebenen zu seinen gewünschten Dokumenten »durchklicken« muss. Das größte Problem aber ist, dass diese Kataloge aufgrund ihrer manuellen Wartung sehr unvollständig und sehr wenig aktuell sind.

### 3.2.2 Daten und Metadaten

Ein Weg, die Suchmöglichkeiten im Web qualitativ entscheidend zu verbessern, ist, Ressourcen mit sogenannten *Metadaten*, also mit zusätzlichen Daten über sich selbst oder andere Ressourcen, anzureichern.[3] In gewissem Umfang ist dies seit den Anfängen des WWW Praxis: Der Kopf eines HTML-Dokuments kann sogenannte `META`-Tags enthalten, die zusätzliche Informationen über das Dokument – wie z. B. Name des Autors, Beschreibung und Schlüsselwörter – beinhalten. Diese Angaben können Suchmaschinen ermöglichen, Dokumente auch anhand von Suchbegriffen zu finden, die im Doku-

[2] `http://www.yahoo.de`

[3] Genau genommen sind natürlich auch Auszeichnungen in XML-Sprachen *Metadaten*. Um Missverständnisse zu vermeiden wird der Begriff hier aber nur auf Ressourcenebene verwendet.

```
<META name="keywords"
        content="uni.xsl, Template, Anpassung, xslt">
<META name="description"
        content="Anleitung zur Anpassung des
                 Uni-Standard-Templates">
```

**Beispiel 3.2.1:** META-Informationen in HTML-Dokumenten

ment selbst nicht vorkommen, indem z. B. das Metaattribut Keywords entsprechend gesetzt wird (siehe Beispiel 3.2.1).

Leider ist dieses Verfahren inzwischen praktisch nur noch für lokale Suchmaschinen einsetzbar, die auf Inhalten einer einschätzbaren Vertrauenswürdigkeit operieren. Denn viele, insbesondere kommerzielle Anbieter von Webseiten sind natürlich längst dazu übergegangen, Schlüsselwörter nicht mehr zu einer adäquaten Beschreibung ihres Angebots zu verwenden, sondern allein zu dem Zweck, bei möglichst vielen Suchmaschinen-Anfragen möglichst weit oben auf der Trefferliste zu erscheinen. Dieses sogenannte *keyword spamming* hat dazu geführt, dass globale Suchmaschinen Keywords inzwischen entweder vollständig ignorieren oder sie nach geheimgehaltenen Algorithmen filtern und gewichten.

Um in Zukunft die Auffindbarkeit von Informationen im Web zu verbessern, erscheint es deshalb notwendig, zum einen die Konfidenz in Metainformationen kalkulierbarer zu machen oder bzw. und auf der anderen Seite, ein erfolgreiches Spamming durch spezifischere Metadaten und damit einhergehenden spezifischeren Suchmöglichkeiten zu erschweren.

Neben der oben angesprochenen Wertlosigkeit von Metainformationen für die Auffindbarkeit von Informationen in globalen Suchmaschinen weist die klassische Angabe von Metainformationen im Kopf von HTML-Seiten weitere Einschränkungen auf, die ihrer Brauchbarkeit im Hinblick auf ein Netz von automatisch verarbeitbaren Informationen im Wege stehen. Zum einen ist die Menge möglicher Attribute vorgegeben und so klein, dass eine gezielte Suche nach Attribut-Wert-Paaren außer nach dem Namen des Autors nicht denkbar ist. Zum anderen kann der Bezug der Metadaten nicht näher spezifiziert werden, so dass entweder grundsätzlich nur Angaben zum aktuellen Dokument selbst gemacht werden können oder Aussagen über andere, z. B. im Dokument referierte Objekte (Produkte, andere »Webobjekte«), natürlich-

sprachlich innerhalb des Werte-Arguments gekennzeichnet werden müssen.

Letztendlich resultiert die klassische Angabe und Verwendung von Metadaten etwas überspitzt formuliert in folgendem Dilemma: Entweder Metadaten sind wertlos, weil sie nicht ausreichend genau spezifiziert werden können, oder Metadaten sind wertlos, weil die Attribut-Wert-Paare ebenso wenig gezielt automatisch auswertbar sind wie die Dokumente, die sie beschreiben sollen.

## 3.3 Das Resource Description Framework (RDF)

Mit dem Ziel ein besseres Gerüst für ein Netz aus *maschinenverständlichen* im Gegensatz zu nur *maschinenlesbaren* Daten zu schaffen, hat das W3-Konsortium bereits 1999 das sogenannte Resource Description Framework (RDF) [LS99; AAB+02] als Empfehlung verabschiedet. RDF soll dabei die in den vorausgegangenen Abschnitten dargestellten Probleme überwinden, also eine allgemeine Grundlage für den Austausch und die automatische Verarbeitung von Informationen im Web schaffen, auf Basis derer

- eine bessere Auffindbarkeit von Ressourcen,
- eine Katalogisierung von Inhalten,
- eine automatische Weiterverarbeitung durch intelligente *Softwareagenten*
- und eine Beschreibung von Seitensammlungen die zusammen ein logisches Dokument ergeben

erreicht werden kann. Gleichzeitig soll RDF aber auch Mechanismen bereitstellen, um *intellektuelle Eigentumsrechte* und Präferenzen bezüglich des *Privatheitsgrades von Informationen* festzulegen [P3P02] – was gerade im Hinblick auf ein Netz maschinell verarbeitbarer Informationen besonders wichtig erscheint.

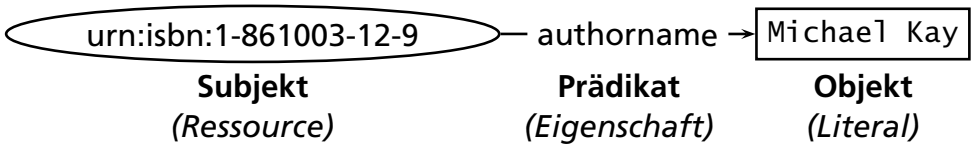


**Abbildung 3.1:** *Einfaches RDF-Modell in Graphendarstellung.*
Ellipsen repräsentieren Ressourcen, Kanten Eigenschaften und Rechtecke Literale.

### 3.3.1 Das RDF-Datenmodell

RDF ist zunächst keine Sprache, sondern ein unabhängig von einer speziellen Syntax definiertes Datenmodell. Das Datenmodell besteht aus drei Objekttypen:

**Ressourcen** Alle Dinge, die sich mit RDF-Ausdrücken beschreiben lassen, werden *Ressourcen* genannt. Es kann sich dabei um Webseiten handeln, aber auch um Teile von Webseiten, einzelne Elemente innerhalb von HTML- oder XML-Dokumenten, komplette Web-Sites und auch nicht direkt über das Web zugängliche abstrakte oder reale Objekte, wie z. B. Videokassetten. Ressourcen werden durch URIs plus optionale Anker-IDs bezeichnet (siehe Abschnitt 1.1.3).

**Eigenschaften** Eine *Eigenschaft* (»*property*«) ist ein bestimmter Aspekt, ein Charakteristikum, ein Attribut oder eine Relation, verwendet zur Beschreibung einer Ressource.

**Aussagen** *Aussagen* (»*statements*«) bestehen aus einer Ressource, einer Eigenschaft und einem Wert. Diese drei Bestandteile werden auch *Subjekt*, *Prädikat* und *Objekt* genannt.

Eine RDF-Aussage zur Repräsentation des einfachen Sachverhalts: »*Michael Kay ist der Autorenname eines durch die ISBN-Nummer 1-861003-12-9 eindeutig bezeichneten Buches.*« würde sich also z. B. aus dem Subjekt (Ressource) `urn:isbn:1-861003-12-9`, dem Prädikat (Eigenschaft): `authorname` und dem Objekt (Literal) `Michael Kay` zusammensetzen. Zur Darstellung von RDF-Mo-

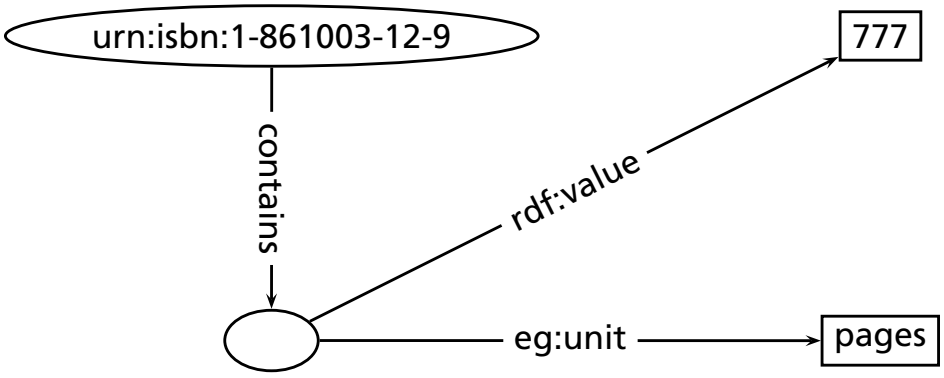


**Abbildung 3.2:** RDF-Aussage mit anonymem Knoten

dellen werden in der Regel gerichtete Graphen mit benannten Kanten verwendet (siehe Abbildung 3.1 ), da diese Koreferenzen unmittelbar sichtbar machen.

Über einfache, wie in Abbildung 3.1 dargestellte, binäre Relationen/ Eigenschaften hinaus lassen sich in RDF aber auch allgemeine *n-äre Relationen* repräsentieren. Eine Möglichkeit, eine mehrstellige Relationen zu repräsentieren, besteht darin, sogenannte *anonyme Knoten* einzuführen, die die an der Relation beteiligten Objekte miteinander verbinden und als existenzquantifiziert interpretiert werden. Dieses Verfahren wird hauptsächlich dann verwendet, wenn Objekte z. B. durch (Maß-)Einheiten näher klassifiziert sind, wie etwa in »*`urn:isbn:1-861003-12-9` enthält 777 Seiten.*« (siehe Abbildung 3.2).

Für Aussagen mit mehrstelligen Prädikaten wird dagegen eine andere Darstellungskonvention gewählt: Die Aussage wird *reifiziert* (»verdinglicht«). Das heißt, Subjekt, Prädikat und Objekt einer Aussage werden zu expliziten Eigenschaften eines anonymen Knotens vom Typ `Aussage`.

Reifikation wird aber vor allem benötigt, um Aussagen über Aussagen machen zu können, also Aussagen höherer Ordnung zu repräsentieren (siehe Abbildung 3.3 ). Diese Möglichkeit ist im Hinblick auf ein Netz, in dem sich die Sicherheit, Qualität oder Vertrauenswürdigkeit von Aussagen ausdrücken bzw. abschätzen lässt, von entscheidender Bedeutung. Mit der Mög-

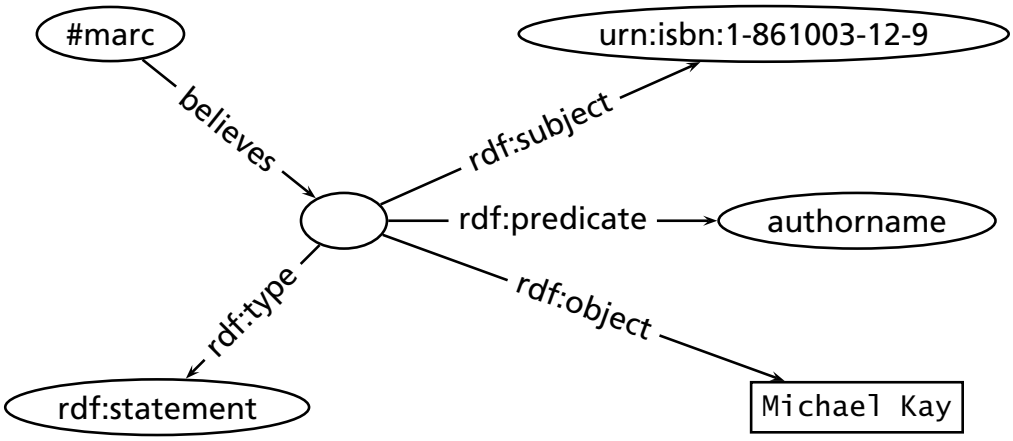


**Abbildung 3.3:** *RDF-Aussage zweiter Ordnung durch Reifikation.*
Durch Reifikation einer Aussage ändert sich ihre Graphenstruktur so, dass sie Objekt einer anderen Aussage werden kann (vgl. Abbildung 3.1 auf Seite 62).

lichkeit der Reifikation verbundene Probleme, wie das Verlassen der *Prädikatenlogik erster Stufe* (PL1) als theoretisch fundierte, handhabbare, logische Grundlage, müssen also ohnehin in Kauf genommen werden. Auch die mit der Reifikation einhergehende starke Veränderung der Modellstruktur einer Aussage (siehe Abbildung 3.3) und die dadurch erhöhte Verarbeitungskomplexität (allein bei der Überprüfung der Äquivalenz zweier Aussagen) muss als Kehrseite des sehr einfachen tripelbasierten Formalismus, der auf der anderen Seite die Repräsentation einfacher Sachverhalte einfach hält, betrachtet werden.

Als weitere Repräsentationshilfen stellt RDF drei sogenannte *Container-Objekte* zur Verfügung, deren genaue Bedeutung nach der zur Zeit aktuellen Version der modelltheoretischen Semantik zu RDF allerdings weniger semantisch als als vielmehr *pragmatisch* restringiert ist:

> »*One should understand this RDF container vocabulary as providing a very weak language for describing containers, rather than as a vocabulary for constructing containers, as would typically be supplied by a programming language. On this view, the actual containers are enti-*

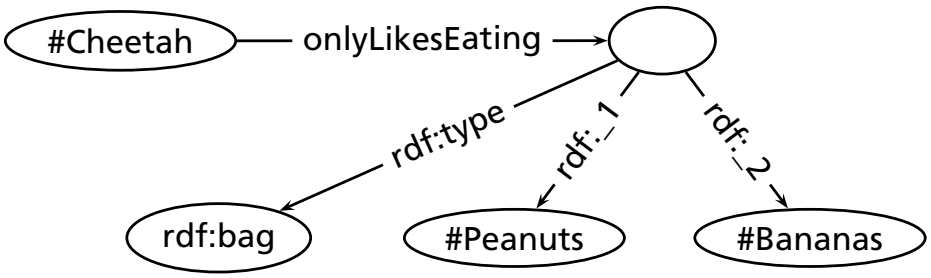


**Abbildung 3.4:** *Verwendung eines RDF-Bag-Container zum Ausdruck von Ausschließlichkeit.*

> *ties in the semantic universe, and RDF graphs which use the container vocabulary simply provide very basic information about these entities, enabling an RDF graph to characterize the container type and give partial information about the members of a container. Since the RDF container vocabulary is so limited, many 'natural' assumptions concerning RDF containers are not formally sanctioned by the RDF model theory. This should not be taken as meaning that these assumptions are false, but only that RDF does not automatically entail that they must be true.*«
>
> Pat Hayes [RDF02a]

Der sogenannte *Alternative-Container* hat die Funktion, alternative Ausprägungen eines Wertes, z. B. verschiedensprachige Übersetzungen eines Dokuments oder verschiedene *Mirrors* eines FTP-Servers, zu repräsentieren. Entgegen der ersten Entwürfe der *RDF Model Theory* ist nach dem aktuellen Entwurf [RDF02a] ein Tripel mit *Alternative-Container* allerdings ausdrücklich *nicht* als Kodierung einer logischen Disjunktion zu verstehen.

Der sogenannte *Bag-Container* kann eine Gruppe von Subjekten oder Objekten zusammenfassen, die mit dem selben Prädikat verknüpft sind. Verwendet wird er dann, wenn eine bestimmte, anonyme Gruppierung der Entitäten ohne eine Festlegung ihrer Reihenfolge zum Ausdruck gebracht werden soll.

Da RDF im Gegensatz zu einigen Wissensrepräsentationssprachen, wie

z. B. PROLOG, nicht von einer *closed world assumption*[4] Gebrauch macht, können *Bag-Container* außerdem dabei behilflich sein *Ausschließlichkeit* zu explizieren. Um z. B. auszudrücken, dass Cheetah ausschließlich Bananen und Erdnüsse gerne isst, könnte man ein Prädikat `onlyLikesEating` zusammen mit einem *Bag*-Objekt verwenden, dass nur Bananen und Erdnüsse »enthält« (siehe Abbildung 3.4 ). Aus der resultierenden, in Abbildung 3.4 dargestellten Aussage kann aber wiederum nicht logisch abgeleitet werden, dass <`Cheetah`, `onlylikesEating`, `Apples`> falsch ist, da die Bag-Container, wie gesagt, nur eine *Bedeutungsabsicht* ausdrücken. Es besteht innerhalb von RDF außerdem keine Möglichkeit, eine Relation zwischen `likesEating` und `onlylikesEating` so festzulegen, dass man aus der Beispielaussage schließen könnte, dass <`Cheetah`, `likesEating`, `Apples`> falsch wäre. Da RDF nur Existenzquantifizierung und Konjunktion, nicht aber Negation und Allquantifizierung beinhaltet, gibt es generell keine Möglichkeit, innerhalb eines RDF-Modells einen Widerspruch zu erzeugen.

Der sogenannte *Sequence-Container* hat die Funktion eine geordnete Liste von Ressourcen oder Literalen zusammenzufassen. Er unterscheidet sich vom *Bag-Container* nur darin, dass eine Reihenfolge zwischen den einzelnen Elementen festgelegt wird.

### 3.3.2 RDF-Syntax

Natürlich wird zur Repräsentation und zum Austausch von RDF-Metadaten auch eine Sprache benötigt, die in der Lage ist, das RDF-Datenmodell auszudrücken, also es in eine serielle Form zu überführen. Zu diesem Zweck wird naheliegenderweise eine XML-Enkodierung – in zwei syntaktischen Varianten – benutzt. In der sogenannten *Basic Serialization Syntax*-Variante lautet die RDF/XML-Repräsentation des einleitenden Beispielsachverhalts folgendermaßen:

```
<rdf:Description about="urn:isbn:1-861003-12-9">
  <ns:AuthorName>Michael Kay</ns:AuthorName>
</rdf:Description>
```

[4] Unter der *closed world assumption* kann, wenn weder p noch ¬p in der Datenbasis vorhanden sind, ¬p der Datenbasis hinzugefügt werden.

Die `ns:`-Präfixe referieren dabei auf einen eindeutig definierten Namensraum. Die Verwendung von Namensräumen ist gerade bei RDF aufgrund der über das Netz verteilten Informationen unerlässlich um sonst vorprogrammierte Namenskollisionen und daraus resultierende Missinterpretationen zu vermeiden.

In der sogenannten *Basic Abbreviated Syntax* werden die in einer Beschreibung enthaltenen Elemente als Attribute in das Beschreibungselement selbst gezogen:

```
<rdf:Description about="urn:isbn:1-861003-12-9">
  ns:AuthorName="Michael Kay"/>
```

Die Frage, die sich angesichts der Beispiele stellt, ist, warum nicht einfach gleich XML zur Kodierung von Metadaten verwendet wird. Ein Grund ist, dass eine direkte XML-Kodierung zu flexibel ist, oder genauer gesagt zu viele Varianten zulässt, ein und dieselbe Sache auszudrücken. Folgende Möglichkeiten sind z. B. denkbar, um ein »rotes Auto« zu beschreiben [Dum00]:

```
<car color="red"/>
<car><color>red</color></car>
<car color="#cc"/><color id="cc" shade="red"/>
...
```

XML erlaubt dabei auch Datenstrukturen, z. B. Mischungen aus Bäumen und Text, die für die Beschreibung des RDF-Datenmodells unnötig komplex sind. Außerdem ist die Reihenfolge der Elemente in XML-Dokumenten signifikant, während eine Reihenfolge-Ordnung von Aussagen in einem Netz aus Metadaten keine Rolle spielt, bzw. nicht besteht. Die RDF/XML-Syntax stellt also eine Konvention dar, die getreu dem Internetmotto *it's not cool to be different* für eine möglichst *einheitliches*, *einfaches* und damit *austauschbares* Repräsentationsformat sorgt. Minimale Komplexität spielt dabei natürlich auch für die Skalierbarkeit im Hinblick auf das Ziel eines globalen (Meta-)Datennetzes – mit möglicherweise vielen Milliarden Aussagen – eine wichtige Rolle.

```
<rdf:Property about="http://www.foo.org/rdf/2002-04/#Author">
 <rdfs:label xml:lang="en">Author</rdfs:label>
 <rdfs:range rdf:resource="http://eg.org/rdf/2002-04/#Person"/>
 <rdfs:domain rdf:resource="http://eg.org/rdf/2002-04/#Document"/>
</rdf:Property>
```

**Beispiel 3.3.1:** *RDFS/XML-Beispiel.*
Die RDF-Schema-Aussage im XML-Notation besagt, dass `eg:Author` eine Eigenschaft ist, die ein Objekt vom Typ `Person` und ein Subjekt vom `Document` verlangt.

### 3.3.3 RDF Schema

Ähnlich wie bei XML ist es auch im Falle von RDF notwendig, Einschränkungen festzulegen zu können und die Integrität von Dokumenten bzw. Modellen anhand dieser überprüfbar zu machen. Anders als bei XML ist XML-Schema jedoch nicht in der Lage diese Aufgabe für RDF zu erfüllen, da im Falle von RDF über einen Datentypintegrität hinaus eine *Ressourcentyp-Integrität* überprüfbar sein muss. Um die Aussage »*`http://eg.org/?id=312` ist Autor von `http://eg.org/welcome.html`.*« auf ihre Plausibilität zu testen, müsste z. B. nicht allein festgestellt werden, ob `http://eg.org/?id=312` ein korrekter URI ist, sondern auch, ob es sich bei der referierten Ressource um eine Person und nicht etwa um eine Webseite handelt. RDF-Schema (RDFS) [RDF02b] erlaubt es unter anderem solche Restriktionen über Ressourcentypen festzulegen. Als Beschreibungsformalismus verwendet RDF-Schema dabei wiederum RDF.

Die Typisierung beruht bei RDF-Schema auf einem einfachen Klassensystem mit folgenden vordefinierten Grundklassen:

**`rdfs:Resource`** Alle Entitäten, die durch RDF-Ausdrücke beschrieben werden können, sind Ressourcen.

**`rdfs:Literal`** Wie oben beschrieben, lässt das RDF-Datenmodell als Objekte nicht nur Ressourcen, sondern auch Literale, also Zeichenketten, zu.

**`rdfs:Class`** ist, vergleichbar mit Klassen in objektorientierten Programmiersprachen, die Teilmenge der Ressourcen, die bestimmte Typen, wie z. B. Personen, Dokumente, Tiere, usw., repräsentieren.

**`rdf:Property`** ist die Teilmenge der Ressourcen, die im Sinne der RDF-Terminologie Eigenschaften repräsentieren, also z. B. hat-Autor, ist-ein, gibt usw.

**`rdf:Statement`** repräsentiert Aussagen über die Eigenschaften von Ressourcen. Wie oben gezeigt, müssen reifizierte Aussagen als `rdf:Statement` typisiert werden.

Um Beziehungen zwischen Klassen herzustellen, stellt RDF folgende Eigenschaften zur Verfügung:

**`rdf:type`** gibt an, dass eine Ressource Mitglied einer Klasse, also Instanz einer Klasse, ist.

**`rfds:subClassOf`** gibt an, dass eine Klasse eine Unterklasse, also Spezialisierung einer anderen Klasse, ist. Im Unterschied zu z. B. Java kann eine Klasse dabei mehrere Klassen spezialisieren. Ein weiterer Unterschied zur klassischen OO-Systemen besteht darin, dass mit der Subklassen keine Vererbung von Eigenschaften einhergeht, da RDF-Klassen selbst keine Eigenschaften aufweisen. Die Unterklassenbeziehung ist vielmehr für die Typen-Einschränkungen und die Inferenz von unbekannten auf allgemeinere, bekannte Klassen wichtig (siehe Abschnitt 3.3.4.3).

**`rdf:subPropertyOf`** zeigt, analog zu `rfds:subClassOf`, die Spezialisierung einer Eigenschaft an.

RDF Schema definiert eine Klasse nicht, wie in klassischen objektorientierten Systemen üblich, anhand seiner Merkmale, sondern umgekehrt, Merkmale anhand der Ressourcenklassen auf die sie anwendbar sind. Wie Beispiel 3.3.1 zeigt, wird `Author` als Merkmal mit der Ausgangsdomäne (`domain`) `Document` und dem Wertebereich (`range`) `Person` definiert, während man in klassischen OO-Sprachen eine Klasse `Document` mit dem Merkmal Author vom Typ *`Person`* definieren würde. Der Vorteil des RDF-Ansatzes ist, dass Benutzer nachträglich weitere Merkmale mit der Ausgangsdomäne `Document` oder

dem Wertebereich `Person` hinzufügen können, ohne dass sie dabei die Klassen durch eine Neu-Definition überschreiben oder unmotivierte Spezialisierungsklassen einführen müssen und selbstverständlich ohne dass sie lokale Kopien der verwendeten bzw. erweiterten Klassen benötigen. So kann Benutzer `B` leicht Eigenschaften zu einer von Benutzer `A` definierten Klasse hinzufügen, deren Notwendigkeit dieser nicht vorhergesehen hat. Diese spezielle Eigenart ist für den Verwendungszweck von RDF-Schema von elementarer Wichtigkeit, da sie eine *dezentrale, sukzessive und Redundanzvermeidende Erweiterung von Ressourcenklassen* ermöglicht und damit gemäß Tim Bernard-Lees Webarchitektur-Prinzips »*anyone being (technically) allowed to say anything about anything*« [BL98b] die Grundlage für über das Web verteilte und über das Web evolvierende Schemata legt.

## 3.3.4 Perspektiven von RDF(S)

### 3.3.4.1 Vokabularien

RDF verfügt also über eine Schemasprache, in der sich, über die Fähigkeiten von XML-Schema hinaus, semantische Restriktionen über referierte (Web-)Objekte formulieren lassen und die vor allem dem Semantic-Web-Ziel einer über das Netz verteilten Informationsrepräsentation Rechnung trägt. Außerdem vermeidet RDF das Problem eines übernotwendig großen syntaktischen Variationsspektrums. Die Frage, die sich jedoch noch stellt, ist, ob RDF das babylonische Verständigungsproblem nicht lediglich eineinhalb Stufen höher auf die Ebene einer Konstituentensemantik verschiebt. Schließlich legt RDF zwar auf der syntaktischen Ebene die kanonische Form zur Beschreibung des Sachverhalts »Das Auto ist rot.« und die auf der semantischen Ebene resultierende Relation fest, aber RDF legt letztendlich nicht fest, was die einzelnen Konstituenten »Das Auto«, »ist« und »rot« bedeuten. Zu dieser Frage ist zunächst anzumerken, dass ein Gerüst zur Repräsentation von Metadaten die Aufgabe, ein universell verbindliches Lexikon oder *Vokabular* – auch *Grand Ordinate Directory* (GOD) genannt – vorzugeben, nicht leisten kann und deshalb dies auch nicht versuchen sollte. Oder wie Tim Bray es ausdrückt:

> *»In fact, there is no www.GOD.org. For this reason, there is no chance that everyone will agree to start using the same metadata facilities. If libraries, which have existed for hundreds of years, can't agree on a single standard, there's not much chance that the Web will. «* [Bra01]

Stattdessen müssen sich die Vokabularien innerhalb der unterschiedlichen Domänen entwickeln, so dass sich mit der Zeit die geeignetsten als domänenspezifische oder domänenübergreifende De-facto-Standards herauskristallisieren.

#### 3.3.4.2 RDF-Anwendungsbeispiele

Ein verbreiteter bereits etablierter RDF-Standards ist z. B. das ursprünglich von Netscape für »MyNetscape« entwickelte *RDF Site Summary*-Format (RSS) [Sch00], das insbesondere zur Zusammenfassung und Verteilung (»*Content Syndication*«) von Nachrichten zur Verwendung in Newstickern genutzt wird.[5] RSS ist zugleich ein Beispiel für eine domänenübergreifende Einigung auf ein RDF-Schema und für die durch *remote inheritance* erzielbaren Synergieeffekte. Zur Repräsentation von Kernmetadaten, wie Autorenschaft, Erstellungsdatum, Sprache usw. greift RSS nämlich auf die RDF-Variante des ebenfalls sehr weit verbreiteten Metadatenstandards der *Dublin Core Metadata Initiative* [KS02] zurück, die zur Repräsentation von Personen-Kontaktinformationen wiederum Verweise auf die RDF-Variante des verbreiteten Standards für elektronische Visitenkarten vCard [Ina01] nahelegt.

RSS zeigt also, dass begründete Hoffnung besteht, dass sich innerhalb von Domänen RDF-basierte Metadatensprachen etablieren können und diese wiederum zu allgemeineren, ausdrucksmächtigeren Sprachen zusammenwachsen. Dieser Effekt, liegt dabei nicht allein in dem natürlichen, webbedingten Homogenisierungsdruck (siehe Abschnitt 3.1) und den bisher fokussierten syntaktischen Eigenschaften von RDF begründet, sondern vor allem in seinen Modelleigenschaften.

[5] RSS Newsticker werden z. B. angeboten von Heise (`http://www.heise.de/newsticker/heise.rdf`), der Tagesschau (`http://www.tagesschau.de/newsticker.rdf`), N24 (`http://www.n24.de/rss/`) oder Slashdot (`http://slashdot.org/slashdot.rdf`).

#### 3.3.4.3 Semantische Interoperabilität

Ein wichtiger Faktor für die Emergenz von allgemein zumindest partiell »verständlichen« RDF-basierten Metadatensprachen ist die Möglichkeit, RDF-Klassen in Hierarchien anzuordnen. Bestehende Klassen können so durch die Bildung von Subklassen verfeinert und erweitert werden. Der dadurch erzielte Effekt besteht aber nicht nur – wie bei klassischen OO-Sprachen im Wesentlichen der Fall – darin, dass Bestehendes wiederverwendet werden kann und so das Rad nicht jedes Mal neu erfunden werden muss. Zum einen werden im Falle von RDF bestehende Schemata weniger wiederverwendet als tatsächlich *geteilt*. Die Wichtigkeit dieser Eigenschaft transzendiert dabei die üblichen Vorteile einer Redundanzvermeidung, da eine *Up-to-Date*–Haltung und Synchronisation aller Komponenten von, das *World Wide Semantic Web* (WWSW) umspannenden, Klassenhierarchien nicht möglich wäre. Zum anderen – und das ist der im Hinblick auf semantische Interoperabilität entscheidende Faktor – können Softwareagenten auch partiell solche Klassen verarbeiten, die sie nicht kennen, wenn auf bekannte übergeordnete Schemata geschlossen werden kann. Hilfreich hierbei ist auch, dass Klassen auf mehrere Superklassen verweisen können, da Softwareagenten so mehrere Wege eröffnet werden können, auch aus unbekannten Schemata verwertbare Informationen zu erschließen. Dass dabei nicht alle RDF-Aussagen in ihrer vollen Bedeutung erfasst werden können, ist nicht zwangsläufig problematisch, da in der Regel Softwareagenten auch nur solche Aussagen verstehen müssen, die für ihre Aufgabe relevant sind.

Ziel ist also *nicht* eine Metadatensprache, die vollständig von jedem Teilnehmer am WWSW verstanden wird, sondern nur eine Metadatensprache deren domänenspezifische Vokabularien innerhalb der entsprechenden Domänen verstanden werden. Eine graduelle und damit robuste semantische Interoperabilität *kann* also auf Basis von RDF(S) erreicht werden. Dafür dass dieses Ziel tatsächlich erreicht wird, sprechen – noch einmal zusammengefasst – folgende Punkte:

1. Metadaten, die niemand versteht sind wertlos. Deshalb zahlt es sich aus, auf vorhandenen De-facto-Standards aufzubauen.

2. Es ist ökonomischer, bestehende Schemata zu verwenden als Schemata von Grund auf neu zu entwickeln.

3. Verfeinerungen und Erweiterungen bestehender Schemata sind auch für Agenten verwertbar, denen nur die übergeordneten Schemata bekannt sind.

#### 3.3.4.4 Die Rolle von RDF(S) im Semantic Web

In den vorangegangenen Abschnitten wurde im Zusammenhang mit RDF von computerverständlichen und von Softwareagenten verwertbaren Informationen gesprochen. Wie genau sieht diese *Verwertbarkeit* aus und welche Arten von Aussagen lassen sich mithilfe von RDF überhaupt repräsentieren?

Vergleicht man RDF(S) in seiner Ausdrucksmächtigkeit mit der Prädikatenlogik erster Stufe fällt auf, dass auf der einen Seite Negation, Disjunktion, allgemeine Implikation und Allquantifizierung nicht repräsentierbar sind; auf der anderen Seite aber, über PL1 hinaus, Aussagen höherer Ordnung gemacht werden können. Außerdem fällt auf, dass für RDFS keine vollständige Semantik existiert. Eine vollständige Semantik für RDF ist aber auch nicht möglich bzw. kann nicht intendiert sein, da eine solche entweder die *pragmatische* Ausdruckskraft in inakzeptablem Maße reduzieren würde, weil z. B. auf Container-Objekte oder Aussagen höherer Ordnung verzichtet werden müsste oder die Modellkomplexität in inakzeptablem Maße wachsen würde, da z. B. die Formalisierung von Bag-Containern (siehe Abschnitt 3.3.1) die Existenz von Widersprüchen ermöglichen würde.

Das Fehlen einer formalen Semantik für RDFS war häufig ein Gegenstand der Kritik, betrachtet man RDF jedoch in Hinblick auf seine intendierte Rolle, nämlich die einer minimalen Grundlage für das *Semantic Web* ist seine Einfachheit der konkreten Entstehung eines *Semantic Web* mehr zuträglich als seine Nachteile dieser Entstehung hinderlich sind. So ist das in seiner Bedeutung nicht scharf festgelegte Container-Modell sicherlich kein Hemmnis, sondern, ebenso wie Vokabularien, Gegenstand und darüber hinaus Katalysator einer – unter Umständen zunächst domänenspezifischen bzw. Interessengruppen-spezifischen – *Konventionsbildung*. Die Bildung von RDF(S)-*Sprachkonventionen* (im Sinne von David Lewis [Lew69]), analog zur Bildung

natürlichsprachlicher Konventionen, ist aufgrund des distribuierten und robusten Charakters von RDF(S) (innerhalb von Interessengemeinschaften) durchaus wahrscheinlich. In diesem Sinne könnte das *Semantic Web* auf der Ebene (s.u) von RDF(S) den Charakter einer Art »*Pragmatic Web*« bekommen.

Der für den Erfolg des *Semantic Web* grundlegende und entscheidende *Netzwerkeffekt* beruht in der durch RDF avisierten *semantischen Schicht*[6] darauf, dass das über das Netz verteilte Wissen über eine bestimmte Ressource ausgehend von der Ressource selbst zugänglich ist. Dies allein entspricht in etwa Bernard-Lees ursprünglicher Vorstellung vom WWW, nach der Links zwischen Dokumenten in beide Richtungen verfolgbar sein sollten – was allerdings aus logistischen Gründen nicht möglich war und nur über den Umweg einer Suchmaschine imperfekt simuliert werden kann. Über diese Vorstellung hinaus können mit Hilfe von RDFS aber nicht nur Beziehungen zwischen Dokumenten bzw. Objekten im Web hergestellt werden, sondern Beziehungen zwischen beliebigen Entitäten. Zudem sind diese Beziehungen nicht wie Links in HTML-Dokumenten anonym, sondern in ihrer Bedeutung eindeutig definierbar, so dass auf der Suche nach einer bestimmten Information über eine bestimmte Ressource nicht alle von dieser ausgehenden Pfade im (semantischen) Netz verfolgt werden müssen und – dank der durch Softwareagenten interpretierbaren Semantik von Eigenschaften bzw. Relationen – nicht alle Zwischenstationen auf dem Weg zu der gesuchten Information *besucht* werden müssen. Logische Implikationen spielen auf dieser Ebene des Semantischen Netz nur insofern eine Rolle, als dass die Bedeutungen von Relationen, ebenso wie die von Objekten, in eine Spezialisierungshierarchie eingeordnet werden können, innerhalb derer Softwareagenten von unbekannten auf bekannte Relationen bzw. Objektklassen schließen können.

[6] Oberhalb dieser *semantischen Schicht* sind weitere Schichten (und entsprechende Formalismen) in Entwicklung: eine *ontologische Schicht*, die je nach Formalismus (z. B. OIL [LB02]) weitere Modellierungsprimitive (z. B. Äquivalenz, Transitivität, ...) beinhaltet, sowie sogenannte *Logic-* und *Proof-Layers*. Die genannten Schichten gehen allerdings über den Skopus dieser Arbeit hinaus und werden im Weiteren zumindest nicht direkt behandelt.

### 3.3.5 Automatische Generierung von RDF-Aussagen

Eine der größten Hürden auf dem Weg zu einem *Semantic Web* ist das klassische »Henne-Ei«-Problem: Auf der einen Seite wird der Aufwand einer RDF-Auszeichnung von Inhalten nur in Kauf genommen, wenn dem Aufwand auch ein Nutzen gegenübersteht. Auf der anderen Seite ist ein *Semantic Web* erst dann nützlich, wenn eine kritische Masse an RDF-Aussagen vorhanden ist.

Eine vielversprechende Option, wie zumindest diese kritische Masse erreicht werden kann, ist, RDF-Metadaten nicht aufwendig von Hand zu erzeugen, sondern sie automatisch zu generieren. Analog zu den in Kapitel 2 beschriebenen Techniken zur Aufbereitung von generisch annotierten Inhalten für das *World Wide Web*, können diese ganz im Sinne der *Single-Source-Publishing*-Idee zumindest teilweise auch für das *Semantic Web* aufbereitet werden. Denkbar sind z. B. auf bestimmte XML-Dokumentklassen spezialisierte XSLT-Scripts, die für das *Semantic Web* relevante Informationen extrahieren und diese nach RDF/XML umwandeln. Bei dokumentorientierten Inhalten liegt z. B. die Transformation von Metadaten in eine Spezialisierung des Dublin-Core-RDF/XML-Formats oder die Generierung von RDF-*Site-Summaries* (siehe Abschnitt 3.3.4.2) nahe. Besonders interessant ist auch die Aufbereitung stark strukturierter datenorientierter Inhalte, insbesondere dann, wenn keine RDF-Aussagen über Entitäten im Web, sondern über Dinge in der Welt generiert werden können. Denkbar sind z. B. semantisch interoperable Produktkataloge, personenbezogene Daten im vCard/RDF-Format oder Veranstaltungskalender im iCalendar/RDF-Format (siehe *RDF Calendar Workspace*[7] bzw. [DS98]). Konkrete Anwendungsbeispiele werden in Abschnitt 5.6.2 dargestellt.

[7] `http://www.w3.org/2002/12/cal/`

# 4 Organisation und Aufbereitung von Inhalten in integrierten Softwaresystemen

Seit Beginn des World Wide Web konnten Autoren Softwaretools in Anspruch nehmen, die die an der Produktion von Webinhalten beteiligten Prozesse unterstützen. Die ersten Hilfen dieser Art waren auf der einen Seite sogenannte WYSIWYG-HTML-Editoren, die es erlaubten Inhalte in gewohnter Weise für das neue Medium zu produzieren und auf der anderen Seite Exportprogramme, mit denen eigentlich für den Druck vorgesehene Dokumente für das Web konvertiert werden konnten.

Durch das so ermöglichte schnelle Wachstum der für das World Wide Web verfügbaren Informationen wird es aber auch immer notwendiger, nicht nur Produktionsprozesse zu unterstützen, sondern auch die immer unüberschaubarer werdende Menge an Inhalten effektiv zu organisieren und zu verwalten, um ihre Wartbarkeit, Aktualität und Funktionalität zu gewährleisten.

Im Folgenden soll beschrieben werden, wie an der Produktion, Verwaltung und Verfügbarmachung von Inhalten beteiligte Prozesse durch Softwaresysteme, sogenannte *Content-Management-Systeme* (CMS), automatisiert werden können. Dabei soll nicht von vorhandenen Softwarelösungen ausgegangen werden, sondern sollen, aufgrund der in den vorangegangenen Kapiteln dargelegten theoretischen Grundlagen, Notwendigkeiten und Prinzipien, und nicht zuletzt den in Kapitel 5 beschriebenen praktischen Erfahrungen, generelle Anforderungen gestellt werden. Anhand dieser Anforderungen werden außerdem Lösungsstrategien entworfen.

## 4.1 Unterstützung und Verwendung von Standards

Eine grundlegende Metaanforderung an Softwaresysteme, insbesondere an solche, die *Inhalte* im Allgemeinen verwalten, ist, dass diese auf frei verwendbaren und allgemein anerkannten Standards beruhen.

### 4.1.1 Vermeidung von Vendor-Lock-Ins

Um das Risiko sogenannter *Lock-Ins*, also Abhängigkeiten von Produkt oder Anbieter, zu minimieren, müssen CMS so weit wie möglich auf *offenen Standards* basieren. Im WWW- und Internetkontext bedeutet das zur Zeit, dass insbesondere die wichtigsten vom W3-Konsortium `http://www.w3.org/` und von der *Internet Engineering Task Force* (IETF[1]) empfohlen Datenformate, Sprachen, Schnittstellen und Protokolle sowie Standard-Programmiersprachen unterstützt werden müssen. Durch die große Menge an z.T. auch freien Werkzeugen und Softwareprodukten, die diese meist sehr ausgereiften Standards unterstützen, kann nicht nur die Einschränkung auf bestimmte Produkte und damit letztlich eine Abhängigkeit von diesen vermieden werden, sondern auch eine optimale *Interoperabilität* zwischen CMS und anderen Produkten erzielt werden.

### 4.1.2 Vermeidung personeller Abhängigkeiten

Die Verwendung von Standardtechniken in CMS ist aber auch im Hinblick auf seine Anwender sinnvoll. Autoren, Designer, Webmaster und Softwareentwickler sind stärker motiviert, sich solche Techniken anzueignen, die ihnen für den weiteren Verlauf ihrer Karriere wichtig erscheinen, als solche, die keine Rolle mehr spielen werden – selbst wenn Letztere zunächst leichter zu erlernen sein sollten. Außerdem ist nicht nur das Angebot an Software größer, das mit Standardformaten umgehen kann, sondern auch das Angebot an Entwicklern. Letztlich können also durch die Unterstützung von Standards nicht nur Produkt-, sondern auch personelle Abhängigkeiten vermieden werden.

[1] `http://www.ietf.org/`

### 4.1.3 Standards als Basistechnologien

Über die reine *Unterstützung* von Standards – ein Begriff der weit gefasst sein kann – hinaus, sollten CMS möglichst auch auf diesen basieren, d. h. nicht nur entsprechende Schnittstellen zur Verfügung stellen, sondern Standards auch intern verwenden. Wenn ein System z. B. in Java implementiert ist und weitgehend auf Standardklassen (z. B. zum Parsen und Transformieren von XML-Dokumenten) aufbaut, kann an vielen Stellen automatisch von Weiterentwicklungen Dritter profitiert werden. Im Idealfall kann ein CMS so auch ohne Zutun des Anbieters auf einem aktuellen Stand bleiben und selbst nach Ende des Lebenszyklus des Anbieters weiter genutzt werden. Die Nachhaltigkeit eines Systems ist also in starkem Maße von einem soliden, also einem auf Standards basierenden, Fundament abhängig.

### 4.1.4 XML-Unterstützung

Viele CMS-Anbieter werben mit dem Schlagwort »XML-Unterstützung«. Da die Speicherung und Auslieferung von Inhalten – abgesehen von der Angabe des entsprechenden MIME-Typen im HTTP-Antwort-Kopf – unabhängig von Format und Kodierung der Inhalte ist, ist eine reine *Unterstützung* ohne nähere Angaben in diesem Sinne trivial. Im Folgenden soll deshalb unter *XML-Unterstützung*, insbesondere die in Kapitel 2 thematisierte Verarbeitbarkeit von XML-Dokumenten und die Unterstützung der in Kapitel 1 vorgestellten Schnittstellen zum Zugriff auf XML-Dokumente verstanden werden.

## 4.2 Protokolle

Um auf die im CMS gespeicherten Daten zugreifen zu können, ohne dabei auf systemeigene Schnittstellen eingeschränkt zu sein, müssen CMS bestimmte Standard-Kommunikationsprotokolle unterstützen.

### 4.2.1 HTTP und HTTPS

Die Unterstützung des *Hypertext Transfer Protocol* (HTTP 1.0) [BLFF96] ist natürlich die Voraussetzung zum Betrieb eines Webservers. Um ein korrektes Caching von ausgelieferten Seiten durch Proxy-Server (siehe Abschnitt 4.4.2) und Browser zu erlauben, sollten CMS dabei insbesondere auch die HTTP-Header-Variablen zum Zeitpunkt der letzten Modifikation eines Dokuments (`last-modified`) und zum Verfallsdatum eines gecachten Dokuments (`expires`) ausliefern. Die Berechnung der Änderungsdaten ist dabei unter Umständen nicht trivial, da durch die Verwendung von Stylesheets auszuliefernde HTML-Seiten sich aus unterschiedlich aktuellen Quellen zusammensetzen können. Idealerweise sollte ein CMS Änderungsdaten automatisch berechnen, aber auch durch Stylesheets explizit setzen lassen. Die Default-Gültigkeitsdauer von Ressourcen sollte möglichst anhand des Ressourcentyps konfigurierbar sein und ebenfalls durch explizite Angaben in Stylesheets oder Metadaten überschreibbar sein.

Um auch ein nicht serverseitiges Cachen (siehe Abschnitt 4.4.2) von Ressourcen zu erlauben, bei denen ein URI mehrere – z. B. von der Sprachpräferenz des Benutzers, vom Benutzeragenten oder vom autorisierten Benutzer abhängige – Versionen eines Dokuments adressiert, muss ein CMS außerdem das erst mit der HTTP-Version 1.1 [FGM+99] eingeführte `vary`-Feld im HTTP-Header unterstützen. Mit Hilfe dieses Feldes kann Proxy-Servern oder Browsern mitgeteilt werden, von welchen Variablen im Kopf der HTTP-Anfrage ein ausgeliefertes Dokument abhängig ist und damit, unter welchen Variablenbelegungen ein gecachtes Dokument wiederverwendet werden kann. Die Angabe

```
Vary: Accept-Language
```

im Kopf einer HTTP-Antwort besagt z.B., dass das angefragte Dokument abhängig von den im Anfragekopf angegebenen Sprachpräferenzen des Benutzers variieren kann.

Damit vertrauliche Informationen sicher – das heißt verschlüsselt – zwischen Client und Server übermittelt werden können, sollte der einem CMS zugrundeliegende Webserver außerdem die auf dem sogenannten *Secure*

*Socket Layer* (SSL) operierende HTTP-Variante HTTPS unterstützen.

### 4.2.2 WebDAV

Bei WebDAV (*Web based Distributed Authoring and Versioning*) [GWF+99] handelt es sich um eine Erweiterung zu HTTP/1.1, die speziell zum *Web Authoring* benötigte Formate und Methoden definiert. Im Einzelnen erweitert WebDAV HTTP/1.1 um Operationen auf:

- *Ressourceneigenschaften* (»*properties*«) bzw. *Metadaten*: Ressourceneigenschaften können gesetzt, verändert, erfragt oder gelöscht werden. Die Eigenschaftswerte können in Form einfacher Zeichenketten und in XML-Form vorliegen. Bei Eigenschaften wird unterschieden zwischen sogenannten *live properties*, deren Syntax und Semantik serverseitig definiert ist (z. B. Modifikationsdatum) und sogenannten *dead properties*, die vom Client frei definiert werden können und vom Server lediglich gespeichert werden.

- *Sammlungen* (»*collections*«): Ressourcen können zu Sammlungen zusammengefasst werden. Hierarchische Mitglieder-Listings solcher Sammlungen können (wie Directory-Listings von Dateisystemen) abgefragt werden.

- *Locking*: Ressourcen können, während sie von einer Person bearbeitet werden, für den Schreibzugriff von anderen gesperrt werden (vgl. Abschnitt 4.7.1).

- *Versions-kontrollierten Ressourcen* (WebDAV-Erweiterung[2]): Management von Ressourcen-Versionen, Versions-kontrollierten Ressourcen und Versionsgeschichten.

Die durch den WebDAV-Standard definierte Schnittstelle zum Zugriff auf Meta- und Versionierungsdaten ist für das Content-Management *prinzipiell* von großer Bedeutung (siehe Abschnitt 4.5 bzw. Abschnitt 4.7). In Bezug auf

[2]Die Definition eines Versionskontroll-Protokolls war ursprünglich Bestandteil von WebDAV, wurde aus Komplexitätsgründen jedoch zunächst abgespalten und erst im März 2002 von der IETF verabschiedet [CAE+02].

*De-facto*-Interoperabilität ist mit der serverseitigen Implementation von WebDAV durch ein CMS *zur Zeit* allerdings wenig gewonnen. Das liegt daran, dass WebDAV-Client-Funktionalitäten unter *Windows* und *Mac OS* praktisch nur über kommerzielle Softwareprodukte, wie *Microsoft Office* (aber z. B. auch *Adobe GoLive*) zugänglich sind. Die Funktionalitäten sind dabei außerdem auf diejenigen beschränkt, die bereits FTP (s.u.) zur Verfügung stellt. Das Anzeigen und Setzen von Metadaten wird ebenso wenig unterstützt wie die Versionierungserweiterung.

### 4.2.3 Dateitransfer

Neben der HTTP-Unterstützung ist Zugänglichkeit der vom CMS verwalteten Ressourcen über eine betriebssystemübergreifend verbreitete, frei zugängliche Dateitransfer-Schnittstelle die zweite notwendige Voraussetzung für die Verwendbarkeit eines CMS. Die Implementation einer solchen Schnittstelle *im CMS*, wird insbesondere dann notwendig, wenn das CMS seine Inhalte auf einer Abstraktionsschicht oberhalb eines konkreten Dateisystems verwaltet (z. B. in einer Datenbank oder einem CVS[3]) und damit die Verwendung der eigentlich in der Verantwortung des zugrundeliegenden Betriebssystems liegenden Möglichkeiten zum *File System Sharing* oder zum Dateitransfer nicht adäquat ist.

Da WebDAV, wie oben dargelegt, zur Zeit nicht ausreichend verbreitet ist, kommt für den Austausch von Dateien zwischen Benutzer und CMS im Wesentlichen nur das sehr weit verbreitete *File Transfer Protocol* (FTP) [PR85], bzw. seine verschlüsselte Variante SFTP in Betracht. Um auch ohne WebDAV nicht nur auf Dateien selbst, sondern auch auf deren Metadaten interoperabel zugreifen zu können, muss ein CMS diese Möglichkeit in die Implementation seines FTP-Servers integrieren. Denkbar ist z. B. die Kennzeichnung von Metadaten durch spezielle Extensionen (z. B. `index.html.meta`). Dieses Feature ist unter Umständen die Voraussetzung für die Erfüllung der wichtigsten Anforderung an ein CMS, nämlich dass alle eingestellten Daten möglichst ohne Systemkenntnis in ihrer ursprünglichen Form wieder exportiert werden können, so dass eine bidirektionale Migrierbarkeit von Inhalten zumindest auf

[3] siehe Abschnitt 4.7.4

der reinen Datenebene nicht eingeschränkt ist.

Um direkt von Anwendungen (z. B. Editoren) und Standardwerkzeugen (z. B. »Suchen/ Ersetzen«) auf CMS-Inhalte zugreifen zu können, ist zusätzlich zu FTP evtl. eine Unterstützung von Protokollen wie SMB oder NFS wünschenswert, die es erlauben die CMS-Inhalte als virtuelles Laufwerk in das eigene Betriebssystem einzubinden. Allerdings wird inzwischen auch das Laden und Speichern von Dateien per FTP von den meisten gebräuchlichen HTML-/ XML- und sonstigen Editoren unterstützt. Außerdem können auch FTP-Server, mit geeigneter Software als Dateisysteme bzw. als Pseudo-Laufwerke eingebunden werden.

### 4.2.4 Protokolle zum Zugriff auf Verzeichnisdienste

Um z. B. zur Authentifizierung eines Benutzers auch auf bereits bestehende Benutzerverzeichnisse zugreifen zu können und damit ein sogenanntes *single-sign-on* zu ermöglichen, sollte ein CMS clientseitig ein entsprechendes Protokoll zum Zugriff auf Verzeichnisdienste unterstützen. Größte Interoperabilität in heterogenen Umgebungen verspricht dabei zur Zeit das *Lightweight Directory Access Protocol* (LDAP) [WHK97; Gre01].

## 4.3 Trennung von Form und Inhalt

Eines der größten Mankos herkömmlichen Web-Publishings war die unzureichende Verteilbarkeit von Aufgaben auf dafür qualifizierte Personen. Waren Autoren selbst für die grafische Gestaltung ihrer Inhalte verantwortlich, mussten sie sich Kenntnisse über Satz, Layout und HTML-Programmierung aneignen. Daraus resultierte nicht nur ein hoher Aufwand bei der Publikation eines Textes im Netz, sondern vielmehr eine Hürde, die häufig so hoch war, dass Texte überhaupt nicht ihren Weg ins Netz fanden. Wenn sie es dennoch taten, waren Mängel in der grafischen Gestaltung, sei es aus unzureichenden Layout- oder HTML-Kenntnissen, vorprogrammiert.

Der einzige Ausweg aus dem Dilemma, der meist auch nur bei kommerziellen Webseiten in Frage kam, war, die Inhalte nachträglich von Webdesignern bearbeiten zu lassen. Dies führte nicht nur zu einer erheblichen Zeit-

verzögerung, sondern auch zu erheblichen Kosten. Zumeist waren auch die Folgekosten hoch, weil sich der fertiggestaltete Webauftritt durch die Autoren selbst nicht mehr ohne Weiteres verändern oder erweitern ließ, da aufgrund von Browserinkompatibilitäten häufig lieber gleich Grafiken als potentiell nicht wunschgemäß dargestellter HTML-Code verwendet wurde, so dass für Änderungen und Erweiterungen des Webauftritts erneut Webdesigner beauftragt werden mussten.

Ein weiteres Problem stellte sich, wenn die Gestaltung des Webauftritts verändert werden sollte. Alle auf die visuelle Form bezogenen Informationen auf jeder einzelnen HTML-Seite mussten verändert werden.

Die Lösung dieser Probleme wird als zentrale Anforderung an ein Content-Management-System betrachtet. Ein CMS muss dazu eine, wie in Kapitel 1 beschriebene *strikte Trennung von inhaltlichen und gestalterischen* Informationen ermöglichen. Das heißt, dass Informationen inhaltlicher und struktureller Art auf der einen Seite und Informationen bezüglich der Darstellung auf der anderen Seite, nicht mehr vermischt, z. B. innerhalb einer HTML-Datei, gespeichert werden. Stattdessen werden Dokumente und Daten getrennt von den Layoutbeschreibungen, die das Aussehen von Dokumenten bestimmen, vorgehalten und für die Generierung von Ansichten auf die Daten, z. B. also für die Ansicht in einem Webbrowser oder zur Ausgabe auf einen Drucker, wieder zusammengefügt.

Autoren können sich ganz auf ihre Inhalte konzentrieren und zeichnen ihre Dokumente idealerweise nur bezüglich ihrer Struktur aus. Sie legen also z. B. nicht fest, dass eine bestimmte Zeile in »`Arial Bold 18pt`« angezeigt werden soll, sondern zeichnen diese Zeile *generisch* als Kapitelüberschrift aus. Wie diese Kapitelüberschrift später aussehen soll, ist in den Regeln eines oder mehrerer zentral vorgehaltener Stylesheets festgelegt. Falls der Internetauftritt einem kompletten Redesign unterzogen werden soll, muss allein das Stylesheet entsprechend geändert werden und alle Seiten erhalten das neue Aussehen.

Die technische Grundanforderung, die sich aus einer Form-Inhalt-Trennung an ein CMS ergibt, ist, dass es über eine »*XML-Unterstützung*« hinaus Mechanismen zur Verfügung stellt, die aus generisch ausgezeichneten Dokumenten anhand von Layoutbeschreibungen gewünschte Ansichten erzeugen

(siehe Kapitel 2). Um die Aktualität der Ansichten zu gewährleisten, sollte die Transformation dabei entweder erst dann vollzogen werden, wenn eine Ansicht angefordert wird oder jedes Mal dann, wenn sich für eine Ansicht relevante Informationen – inhaltlicher oder gestalterischer Art – verändert haben. Insbesondere bei der Verwendung leistungsfähiger Stylesheetsprachen wie z. B. XSLT wird die Aktualität der Webseiten häufig der Serverperformanz geopfert. In einem sogenannten *Staging-Prozess* werden Änderungen an Inhalt oder Gestaltung nur zu bestimmten Zeiten mit geringer Serverbelastung – z. B. einmal pro Nacht – in das Produktionssystem übernommen. Eine solche Vorgehensweise ist allerdings kaum zu begründen, da – eine geeignete Systemarchitektur und geeignete Caching-Konzepte (siehe dazu Abschnitt 4.4.2) vorausgesetzt – der Performanzgewinn so gering sein sollte, dass er den Aktualitätsverlust normalerweise nicht rechtfertigen kann.

## 4.4 Aufbereitung von Inhalten

Um entkoppelte generische und darstellungsorientierte Informationen zur Ansicht im Browser oder zum Ausdruck wieder vereinen zu können, muss ein CMS für diese Aufgabe geeignete Programmiersprachen unterstützen oder geeignete Programmierschnittstellen zur Verfügung stellen. Wie in Abschnitt 2.7 dargelegt, empfiehlt sich zur integrierten Aufbereitung von Daten- und dokumentorientierten Inhalten eine mehrstufige Architektur, in der zunächst das Ausgangsdokument per XSLT transformiert wird und alle weiteren benötigten Informationen in einem zweiten Verarbeitungsschritt mit Hilfe einer eingebetteten Scriptsprache integriert werden.

### 4.4.1 Zugriff auf dokumentexterne Ressourcen

Neben dem Zugriff auf ein »Ausgangsdokument« muss ein CMS auch den Zugriff auf folgende Datenquellen ermöglichen:

- *CMS-interne Quellen:*
  - *Content*: Stylesheets/Scripts sollten selbstverständlich prinzipiell Zugriff auf alle im CMS eingestellten Inhalte zugreifen können

  - *Verzeichnisstruktur bzw. Objekthierarchie*: zur automatischen Generierung von Navigationselementen, wie z. B. aktuellen Position in der Content-Hierarchie, Sitemaps, Menüs, usw. ohne Redundanzen durch separate, die Content-Hierarchie bzw. Verzeichnisstruktur des CMS abbildende Menüdateien zu erzwingen
  - *automatisch generierbare Metainformationen*, wie z. B. Erstellungs- und Änderungsdaten, Anzahl der Seitenabrufe, Verweise zu anderen Versionen oder Sprachversionen einer Datei, z. B. zur Generierung von separaten RDF-Beschreibungsdateien, Standard-HTML-Header-Metadaten oder direkt für den Benutzer bestimmten Metainformation (z. B. im Fußbereich einer Webseite)
  - *vom Benutzer eingestellte Metainformationen*, wie z. B. Titel, Beschreibung und Schlüsselbegriffe (siehe Abschnitt 4.5)
  - *Benutzerverwaltung*: zur automatischen Generierung von Kontaktinformationen, wie z. B. E-Mail-Adresse oder Telefonnummer

- *HTTP-Request-Header-Informationen:*

  - *Angefragter URI bzw. CGI-Parameter*: zur Verarbeitung von Formularen und um die Anfrage spezieller Ansichten einer Seite, z. B. einer Druckansicht, zu ermöglichen
  - *Kennung des verwendeten Browsers* und die von diesem akzeptierte Formate: zur Generierung browserspezifischer Ansichten, wie z. B. einer reinen Textversion für textbasierte Clients
  - *im Client eingestellte Sprachpräferenzliste*: zur Auswahl adäquater Sprachversion von Dokumenten, Bedienungselementen und anderen von Stylesheets generierten Ausgaben (siehe Abschnitt 4.8)
  - *Benutzername* (sofern dieser sich authentisiert hat): zur Anzeige zusätzlicher Informationen (z. B. Seitenzugriffs-Zähler) und zur Generierung zusätzlicher Bedienungselemente, wie z. B. Schalter mit Hilfe derer Seiteninhalte sich direkt editieren lassen
  - *Cookies*

- *externe Quellen:*

- *externe Datenbanken*
- *externe Webserver*: zur integrierten Einbindung externer Webseiten oder Ressourcen im Allgemeinen
- *externe Verzeichnisserver*: zur Integration von Kontaktinformationen aus externen Verzeichnissen

Während interne Dokumente von XSLT/XPath aus zugänglich sind und HTTP-Anfrageparameter, sowie externe Quellen durch Scriptsprachen abgefragt werden können, existieren zum Zugriff auf CMS-spezifische Quellen, wie Metadaten oder Verzeichnisstrukturen, weder für XSLT noch für üblicherweise verwendete Scriptsprachen geeignete *standardisierte* Schnittstellen. Ein CMS muss diese daher selbst definieren und implementieren, so dass eine vollständige Portabilität der am Aufbereitungsprozess beteiligten Stylesheets und Scripts zur Zeit vermutlich nicht erreichbar ist.

Da es sich sowohl bei Verzeichnisinformationen als auch bei Metadaten im Sinne der Charakterisierung aus Abschnitt 1.2.2 eher um semistrukturierte Daten handelt, bietet sich ihre Repräsentation in Form von XML-Dokumenten an. Deshalb sollte bei der Erweiterung von Schnittstellen auch am XSLT-Prozessor angesetzt werden. Die Standard-XSLT-Schnittstellen zur Außenwelt sind: der aktuelle Eingabestrom über den das Hauptquelldokument eingelesen wird, übergebbare Parameter (siehe Abschnitt 2.3) und die XPath-Funktion `document`, die den Zugriff auf andere XML-Dokumente außerhalb des Hauptquelldokuments erlaubt. Um zusätzliche Datenquellen für XSLT-Stylesheets zugänglich zu machen, muss ein CMS also entweder eine der drei Standardschnittstellen um neue Datenübergabe-Konventionen erweitern oder XSLT/XPath selbst um neue Funktionen erweitern, mit Hilfe derer direkt auf die Datenquellen zugegriffen werden kann.

Für die Übergabe von Metadaten zu Ressourcen und zur Verzeichnishierarchie kann eine relativ gute Portabilität erzielt werden, indem diese Information vom CMS in Pseudo-XML-Dokumenten (siehe dazu auch Abschnitt 4.5) zur Verfügung gestellt werden, auf die Stylesheets standardkonform über die XPath-Funktion `document` zugreifen können. Beispiel 4.4.1 zeigt ein mögliches Beispiel für eine XML-Datei mit Informationen zum Inhalt eines Ordners inklusive einiger exemplarischer Metadaten.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dir>
  <name>universitaet</name>
  <path>/universitaet</path>
  <file>
    <title>Universität</title>
    <name>index.xml</name>
    <path>/universitaet/index.xml</path>
    <created>
      <date><year>2001</year><month>9</month><day>2</day></date>
    </created>
    <last-modified>
      <date><year>2001</year><month>9</month><day>2</day></date>
    </last-modified>
    <description>Homepage der Universität Bielefeld</description>
    <keywords/>
    <template>/uni.xsl</template>
  </file>
  <dir>
    <name>..</name>
    <path>/</path>
    <pseudo-metafile>/pseudo:dir-metadata.xml</pseudo-metafile>
  </dir>
  <dir>
    <name>einrichtungen</name>
    <path>/universitaet/einrichtungen</path>
  </dir>
</dir>
```

**Beispiel 4.4.1:** Eine einfache Pseudo-XML-Datei zur Beschreibung eines Ordners

Vorausgesetzt, dass diese Pseudo-Dateien auch als reale Dateien exportierbar sind, ist die Portierung von Stylesheets auf ein anderes CMS mit XSLT-Unterstützung unter minimalem Aufwand zu leisten. Allerdings ist ein solcher Metadatenexport natürlich nur von bedingtem Nutzen, da vom Benutzer eingetragene Metainformationen ohnehin in die Verwaltung des Ziel-CMS importiert und automatisch generierte Metainformationen ohnehin vom Ziel-CMS zur Verfügung gestellt werden müssen.

### 4.4.2 Caching

Eine grundlegende Voraussetzung für den Einsatz leistungsfähiger Transformationssprachen, wie XSLT, zur dynamischen Aufbereitung von Web-Con-

```
<xsl:template match="dir" mode="current.path">
  <xsl:if test="dir[name='..']">
    <xsl:apply-templates mode="current.path"
      select="document(dir[name='..']/pseudo-metafile)"/>
    <xsl:text> > </xsl:text>
  </xsl:if>
  <a href="path">
    <xsl:value-of select="file[starts-with(name, 'index.')]/title"/>
  </a>
</xsl:template>
```

**Beispiel 4.4.2:** XSLT-Template zur Anzeige des aktuellen Pfads

tent ist, dass die Transformation der Rohdaten nach HTML so schnell erfolgt, dass der Webserver eine zufriedenstellende Performanz erreicht. Dies ist bei umfangreichen Transformationen und vielen Zugriffen keineswegs selbstverständlich.

Eine Möglichkeit, ein CMS bei der Aufbereitung von Seiten zu entlasten, besteht darin, Ergebnisse von Transformationen in einem sogenannten *Cache* (deutsch: »*versteckter Vorrat*«) zwischenzuspeichern und bei einer erneuten Anfrage, nicht eine erneute Generierung der Seite anzustoßen, sondern den Cache-Inhalt wiederzuverwenden. Zwischen dem Besucher einer Seite und einem Webserver liegen unter Umständen eine Reihe solcher Zwischenspeicher: sogenannte *Proxy Caches* (s.u.) und der browsereigene Cache, jedoch werden diese, abhängig von Benutzereinstellungen im Browser, umgangen.

#### 4.4.2.1 Inverse Proxy-Caches

Um unabhängig von Benutzereinstellungen ein Caching zu erzwingen, besteht die Möglichkeit, das CMS hinter einem sogenannten *inversen Proxy* zu betreiben. Im Gegensatz zu einem konventionellen Proxy speichert ein inverser Proxy nicht Ressourcen, die von einer bestimmten Gruppe von Benutzern angeforderten werden, sondern umgekehrt solche, die von einem bestimmten Server ausgeliefert werden. Ein solcher inverser Proxy kann dabei in der Regel unabhängig vom verwendeten Webserver bzw. CMS betrieben werden.

Bei einer solchen Konfiguration gehen alle Anfragen zunächst an den Proxy. Falls dieser eine aktuelle Version der benötigten Ressourcen gespeichert hat,

schickt er sie, ohne den eigentlichen Server zu belasten, zurück. Falls er das benötigte Objekt nicht vorrätig hat, holt er sich die aktuelle Version vom Server, speichert sie, um bei zukünftigen Anfragen selbst antworten zu können und leitet sie an den Anfragesteller weiter.

Auch inverse Proxies können Performanzprobleme jedoch nur zum Teil befriedigend lösen. Um zu gewährleisten, dass ein Proxy auf der einen Seite ausnahmslos die aktuelle Version einer Resource ausliefert, auf der anderen Seite aber möglichst wenig Anfragen an den Webserver gestellt werden, muss er im Voraus möglichst genaue Informationen darüber haben, wie lange ein gespeichertes Objekt gültig ist. Wirkliche sichere Vorhersagen über die Dauer der Gültigkeit sind allerdings – wenn überhaupt – nur bei komplett statischem, also unveränderlichem Content, möglich. Doch selbst wenn der einer Webseite zugrundeliegende Content selbst statisch ist, kann es immer noch sein, dass sich Navigationselemente, z. B. durch neue Archiveinträge oder das Layout, z. B. durch neue Stylesheets, der Seite ändern. Seiten mit langer, genau vorhersagbarer Gültigkeit sind also eine Seltenheit, so dass mit inversen Proxies allein die Belastung eines CMS nicht ausreichend gesenkt werden kann, ohne dabei die Aktualität ausgelieferter Seiten zu gefährden.

Um ein optimales Caching bei gleichzeitig optimaler Aktualität zu gewährleisten, muss ein CMS zusätzlich über interne Caching-Mechanismen verfügen, die CMS-interne Abhängigkeiten besser berücksichtigen können.

#### 4.4.2.2 Probleme durch dokumentexterne Abhängigkeiten

Wie in Abschnitt 4.4 dargelegt, müssen an der Aufbereitung von Inhalten beteilige Scripts und Stylesheets auch auf Quelldokument-externe Daten zugreifen können. Ein Problem, das dabei entsteht ist, dass sich mit diesen erweiterten Zugriffsmöglichkeiten auch die Menge der Abhängigkeiten des Aufbereitungsprozesses wächst und damit ein effektives Caching erschwert wird.

Durch eine, wie in Beispiel 4.4.2 dargestellte, rekursive Konstruktion des aktuellen Pfads in der Verzeichnishierarchie anhand von, wie in Beispiel 4.4.1 auf Seite 88 strukturierten Verzeichnismetadaten, ist z. B. das Ergebnis eines XSLT-Transformationsprozesses nicht mehr allein abhängig von Quell-

dokument und Stylesheet, sondern zusätzlich von den Metadaten *aller* im aktuellen Verzeichnis und aller in übergeordneten Verzeichnissen befindlicher Dateien. Dadurch ist ein gecachtes Transformationergebnis nur so lange verwendbar, bis sich die Metadaten *einer* dieser Dateien ändert.

Weitere Abhängigkeiten ergeben sich, wenn Aufbereitungsprozesse auch von HTTP-Request-Header-Variablen, wie z. B. dem Namen des verwendeten Clients, abhängig sind. In diesem Fall muss für jedes Dokument, für jede Kombination von Belegungen der verwendeten Variablen ein eigenes Transformationergebnis gespeichert werden. Mit zunehmender Anzahl von Versionen eines gecachten Dokuments sinkt dabei der durch Caching erzielbare Zeitgewinn gegenüber dem durch Verwaltungsaufwand bedingten Zeitverlust. Im Extremfall sind alle Cache-Versionen (z. B. durch eine Änderung im Dokument) ungültig, bevor eine gecachte Version wiederverwendet werden kann. Ähnliche Probleme ergeben sich durch die Verwendung von CGI-Variablen.

#### 4.4.2.3 Teilbaum-Caching

Da ein leistungsfähiges und performantes CMS weder auf die Möglichkeit Quelldokument-externer Datenzugriffe aus Stylesheets noch auf ein effektives Caching verzichten kann, muss es über Mechanismen verfügen, die über eine einfache Ablage transformierter Dokumente hinausgehen.

Ein generelles Verfahren XSL-Transformationen effektiver zu cachen besteht darin, die Granularität der gecachten Objekte von kompletten Zieldokumenten auf Teilbäume von Zieldokumenten zu verfeinern. Das heißt, nicht nur komplette Transformationsergebnisse, also Webseiten, zu cachen, sondern auch Teile des Transformationsergebnisses, also Seitenkomponenten, wie z. B. Navigationsleisten, Navigationselemente, Inhaltsbereiche, Seitenköpfe und -Füße, usw. separat zu ihrer Wiederverwendung zwischenzuspeichern.

Bei einer geeigneten Auswahl von solchen Teilbäumen kann dadurch sowohl eine längere Gültigkeitsdauer von Cache-Objekten – durch weniger externe Abhängigkeiten in Bezug auf *ein* gecachtes Objekt – als auch eine höhere Wiederverwendbarkeit der Cache-Objekte – durch ihre Nutzbarkeit über

mehrere Ausgangsdokumente hinweg – erzielt werden.[4] In Bezug auf eine Seite kann so im Idealfall die Anzahl der maximal benötigten Caches vom Produkt auf die Summe über die Größe der Wertemengen der relevanten Parameter reduziert werden.

Innerhalb eines, wie in Abschnitt 2.7 skizzierten, mehrstufigen Aufbereitungsprozesses lässt sich dieses Verfahren allerdings nur auf die erste Stufe, also die XSL-Transformationen anwenden. Denn bei XSLT sind, im Gegensatz zu konventionellen Scriptsprachen, die Transformationen disjunkter Teilbäume voneinander unabhängig, so dass Transformationsergebnisse gecacht werden können, ohne dass dabei interne Verarbeitungsabhängigkeiten berücksichtigt werden müssen (siehe Abschnitt 2.3.2). Der Verzicht auf ein Cachen des zweiten Aufbereitungsschritts ist dabei auch durchaus sinnvoll, da dieser ja gerade zur Aufbereitung und Integration besonders dynamischer Daten dient, die, wie z. B. ein Seitenzugriffs-Zähler, unter Umständen erst zum Zeitpunkt der Auslieferung einer Seite bekannt sind. Eine Beschleunigung des zweiten Aufbereitungsschritts lässt sich außerdem erreichen, indem evtl. vorhandene Scripteinbettungen nicht im Quelltext, sondern wie bei einigen Scriptsprachen (z. B. Perl) prinzipiell möglich, in einem bereits interpretierten oder compilierten, sogenannten Byte-Code abgelegt werden

Damit ein Caching von Teilbäumen des XSL-Transformationsergebnisses sinnvoll angewendet werden kann, müssen außerdem bestimmte Bedingungen erfüllt sein. Über verschiedene Seiten wiederkehrende Komponenten sollten in der funktionalen Struktur des Stylesheets ihre Entsprechung, beispielsweise in Form einer Funktion oder Konstante finden. Außerdem müssen die Parameter, von denen zu cachende Stylesheetfunktionen abhängig sind, klar definiert sein, so dass sich z. B. über Funktionsnamen und Parameterbelegungen ein *eindeutiger* Schlüssel bilden lässt, unter dem Cache-Inhalte abgelegt werden und über den diese – wenn sich keiner der Parameter verändert hat – wieder abgerufen werden können.

Das automatische Auffinden von Teilbäumen der Transformation, die sowohl möglichst homogene Abhängigkeiten haben als auch von möglichst vielen Zieldokumenten geteilt werden, ist ebenso wie die Berechnung von

[4] Letztere Konsequenz ist ein Spezialfall, der dann eintritt, wenn ein Cache-Objekt, z. B. ein Navigationselement, unabhängig vom Quelldokument ist.

Abhängigkeiten auf dieser Ebene sehr aufwendig. Deshalb liegt es nahe, ein explizites Cachen von Teilbäumen aus dem Stylesheet – z. B. durch die Einführung eines speziellen Erweiterungselements, dessen Inhalt abhängig von bestimmten Attributen gepuffert wird – heraus zu erlauben (siehe Abschnitt 5.5.1).

#### 4.4.2.4 Abbildung von Parameterwerten auf diskrete Typen

Ein Ansatz, der das oben dargestellte Teilbaum-Caching ergänzen kann, besteht darin, die Anzahl zu cachender, parameterabhängiger Varianten eines transformierten Teilbaums so weit wie möglich zu reduzieren.

Um z. B. bei Abhängigkeiten von HTTP-Anfrageparametern die Anzahl zu cachender Versionen von Transformationsergebnissen auf ein Minimum zu reduzieren, liegt es nahe, die Werte der Anfrageparameter auf die Menge diskreter Typen abzubilden, die für die Transformation tatsächlich eine Rolle spielen. Abbildung 4.1 skizziert z. B. eine Abbildung von Werten der Variable `HTTP_USER_AGENT` über *reguläre Ausdrücke* auf drei Prototypen, die respektive ältere Browsertypen, reine Textbrowser und sonstige repräsentieren. Wenn ein CMS ein solches Mapping unterstützt, können Stylesheets von Browserklassen abhängige Versionen generieren, ohne dass für jede einzelne Browservariante eine unterschiedliche Version generiert werden muss und damit ein effektives Caching unmöglich wird.

## 4.5 Metadaten

Wie bereits in Abschnitt 3.2.2 ausführlich dargelegt wurde, ist die Auszeichnung von Ressourcen mit Metainformationen für ihre Wiederauffindbarkeit und Wiederverwendbarkeit unerlässlich. CMS sollten deshalb auf der einen Seite Autoren bei der Beschreibung ihrer Ressourcen mit Metainformationen und auf der anderen Seite den Benutzer in der Verwendung oder Ausnutzung der Metainformationen unterstützen.

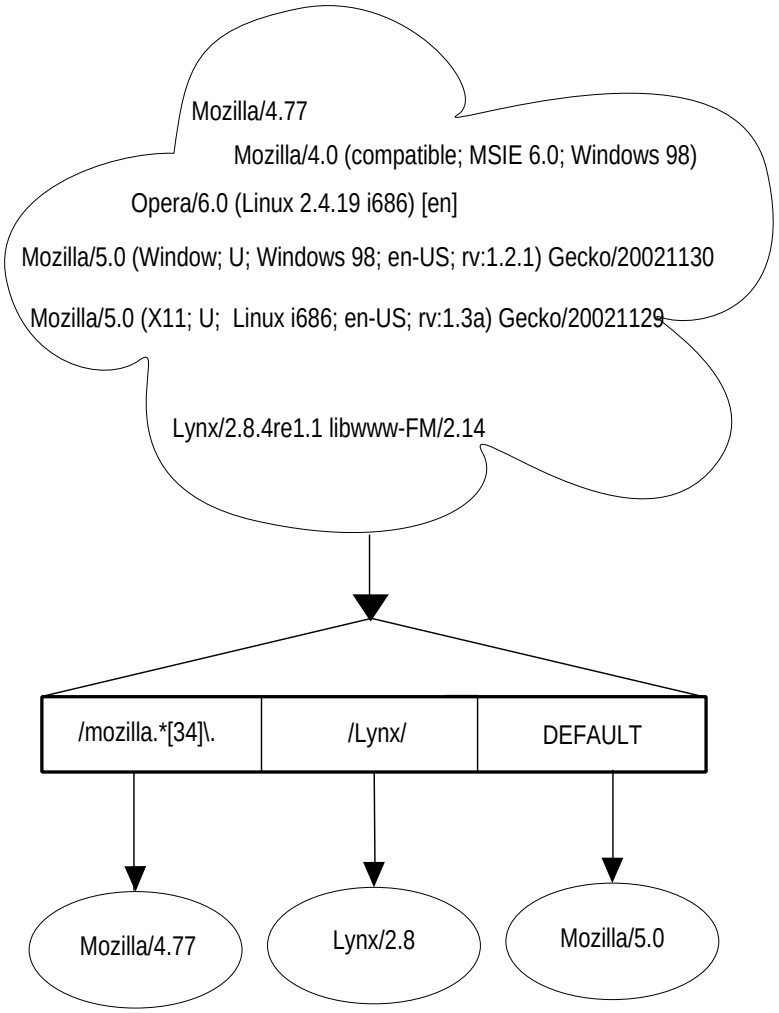


**Abbildung 4.1:** *Abbildung von HTTP-Anfrageparametern auf diskrete Typen anhand regulärer Ausdrücke.*
Um bei Abhängigkeiten von HTTP-Anfrageparametern die Anzahl zu cachenderer Varianten von Transformationsergebnissen auf ein Minimum zu reduzieren, können die Werte der Anfrageparameter (hier *User-Agent*), auf eine Anzahl von Prototypen abgebildet werden.

### 4.5.1 Realisierung im CMS

Um alle Typen von Ressourcen, also z. B. auch Grafiken, mit Metainformationen beschreiben zu können, müssen diese unabhängig von ihrer zugehörigen Ressource zugänglich sein. Da Metadaten außerdem strukturierbar sein sollten, bietet es sich an, diese im CMS wie gewöhnliche XML-Dokumente vorzuhalten. Diese Vorgehensweise hat gegenüber einer speziellen Behandlung eine Reihe von Vorteilen. Zum einen können Metadaten so mit dem großen Pool zur Verfügung stehender XML-Werkzeuge bearbeitet werden. Weiterentwicklungen in Nutzung und Bearbeitung sind nicht von der Weiterentwicklung des CMS abhängig. So kann z. B. die Struktur der Metadaten durch XML- oder RDF-Schemata flexibel definiert werden und Inhalte mit XML- oder RDF-Editoren – falls vom CMS unterstützt – mit XForms verändert werden. Analog zu normalen Dokumenten ist auch die Anwendung von XSLT-Stylesheets auf Metadokumente denkbar, um aus Dokument- und Metadokument auch flexibel externe *Metadatenansichten* – z. B. in verschiedenen RDF-Formaten oder als *Topic Maps* [XTM01; WM02] – (siehe Abschnitt 5.6.1) zu generieren. Die Anforderungen an das CMS sind bei diesem Ansatz auch letztlich gering, da es grundsätzlich nur einen Mechanismus oder eine Namenskonvention zur Verknüpfung von Daten- und Metadaten vorgeben muss und alle anderen Features, wie Editieren, Validieren, Transformieren, Im- und Exportieren ohnehin für den Dokumentkontext vorhanden sein müssen.

Um eine interoperable *funktionale* Verknüpfung von Ressourcen und Metadaten zu erreichen und eine standardisierte Schnittstelle zur Verfügung zu stellen, sollten Metadaten außerdem, wie in Abschnitt 4.2.2 beschrieben, über die HTTP-Protokoll-Erweiterung WebDAV abfragbar und veränderbar sein oder aber zumindest grundsätzlich so strukturiert sein, dass sie zu einem späteren Zeitpunkt prinzipiell per WebDAV verwaltet werden können. Es bietet sich also an, zur Repräsentation von Metadaten direkt die XML-Syntax des entsprechenden WebDAV-*Property*-Elements zu verwenden und auch vorgegebene *Property*-Namen (z. B. `creationdate` [siehe GWF+99]) zu übernehmen. Ein einfaches Metadatendokument könnte z. B. wie in Beispiel 4.5.1 dargestellt lauten.

```
<?xml version="1.0" encoding="utf-8"?>
<d:prop xmlns:d="DAV:" xmls:cms="http://www.mycms.org"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <d:creationdate>2003-02-01T12:00:00Z</d:creationdate>
  <d:displayname>index.html</d:displayname>
  <d:getlastmodified>2003-02-17T23:05:07Z</d:getlastmodified>
  <d:getcontenttype>text/html</d:getcontenttype>
  <d:getcontentlength>124</d:getcontentlength>
  ...
  <cms:stylesheet>/default.xsl</cms:stylesheet>
  <cms:gethits>13</cms:gethits>
  <dc:creator>Marc Kupietz</dc:creator>
  <dc:language>de</dc:language>
  <dc:rights>© Universität Bielefeld</dc:rights>
  ...
</d:prop>
```

**Beispiel 4.5.1:** Ein vereinfachtes WebDAV-konformes Metadatendokument

Wenn solche Metadatendokumente auch über eine FTP-Schnittstelle zugänglich sind, muss das CMS dafür Sorge tragen, dass Veränderungen an schreibgeschützen Eigenschaften (»*dead properties*«), wie z. B. getcontentlength, beim FTP-Upload ignoriert werden.

### 4.5.2 CMS-bezogene Metadaten

Neben modifizierbaren Metadaten (»*live properties*«), die für die Benutzerverwertung bestimmt sind, sind innerhalb eines CMS auch solche notwendig, die festlegen, wie das CMS selbst mit korrespondierenden Ressourcen umgehen soll. Denkbar sind z. B. folgende Angaben:

- Inhaltssprache
- Übersetzungsstatus
- Default-Stylesheet
- Sichtbarkeit nach außen (von, bis)

### 4.5.3 Durchsuchen von Metadaten

Eine große Rolle spielen Metadaten auch für das Wiederauffinden *internen* Contents, also des Contents der tatsächlich im CMS vorhanden ist, im Gegensatz zu dem, der von außen frei zugänglich ist. Ein CMS sollte deshalb autorisierten Benutzern, die Möglichkeit bieten, Metadaten gezielt zu durchsuchen. Für eine Volltextsuche bieten sich dazu Webformulare mit konjunktiv oder disjunktiv verknüpfbaren Wert-Feldern, entsprechend der möglichen Eigenschaften, an. Um auch strukturierte Eigenschaftswerte gezielt durchsuchen zu können, sollte das CMS ein XQuery-Interface (siehe Abschnitt 1.3.2) zur Verfügung stellen, das defaultmäßig über alle (Meta-)Dokumente iteriert und z. B. folgende, teilweise abgekürzte FLWR- bzw. Pfadausdrücke verarbeiten kann:

```
author = "Marc Kupietz" and getlastmodified < "2002-11-26"

contains(keywords, "Studiengebühren")

contains(image-info/visible-persons, "Luhmann")
```

Um diese Suchmöglichkeiten erweitern zu können und sie auch für externe Benutzer flexibel zur Verfügung stellen zu können, sollte das XQuery-Interface in seiner XML-Variante XQueryX (siehe Abschnitt 1.3.2) auch von Stylesheets oder von Dokumenten aus zugänglich sein.

## 4.6 Bild- und Grafik-Management

Um eine möglichst gute Integration von textuellen und grafischen Inhalten zu erreichen und keine Redundanzen durch separate spezialisierte Systeme einzuführen, sollten CMS auch das Management von grafischen Assets unterstützen.

### 4.6.1 Spezielle Anforderungen an Grafik-Metadaten

Da Bilder, Grafiken und Filme selbst nicht auf Schlagwörter hin durchsuchbar sind, nehmen Metadaten – wie bereits an anderer Stelle erwähnt – zur

Sicherung der (Wieder-) Auffindbarkeit von grafischen Inhalten einen besonderen Stellenwert ein. Die für Grafiken relevanten Metadaten unterschieden sich dabei von solchen die für Dokumente relevant sind, bzw. gehen über diese hinaus. Neben Standardmetadaten (z. B. *Dublin Core*) können z. B. folgende relevant sein:

- Informationen zur Art der Grafik (Photo, Zeichnung, ...)
- entsprechende Unterkategorien (Portrait, Gruppenbild, Landschaftsaufnahme, ...)
- abhängig von der Kategorie Informationen technischer Art (verwendete Kamera, Objektiv, Scanner)
- automatisch berechenbare technische Daten (Größe eines Bildes in mm/ Pixeln, Farben, Auflösung, Skalierbarkeit)
- Informationen zum Inhalt (z. B. abgebildete Personen oder Gegenstände)

Die wichtigste Voraussetzung für die sinnvolle Verwaltbarkeit von Bildern innerhalb eines CMS, ist also eine, wie in Abschnitt 4.5 beschriebene, flexible Definierbarkeit von Metadaten und entsprechenden Zugriffsschnittstellen.

### 4.6.2 Grafiktransformationen

Die zweite wichtige Voraussetzung für ein leistungsfähiges Bild-Management ist, dass Bilder und Grafiken, ähnlich wie Dokumente, durch das CMS in verschiedene Ansichten transformiert werden können. Je vielseitiger die Transformationsmöglichkeiten sind, desto umfassender können – gemäß dem Prinzip der Redundanzvermeidung – unnötige Kopien und Ableitungen von Informationen vermieden werden. Wenn ein CMS in der Lage ist, Grafiken *on-the-fly* zu skalieren, verschiedene Formate ineinander zu überführen und Ausschnitte zu extrahieren, reicht es in den meisten Fällen aus, wenn von einer Grafik nur noch eine Version – sozusagen das Original – in maximaler Qualität (z. B. im SVG-, TIFF- oder PNG-Format) gespeichert ist. Alle anderen benötigten Versionen bzw. Ansichten der Grafik, z. B. für den Druck, für

```
<CF_MagickTag action="convert"
   inputtype="file"
   inputfile="pandababy.jpg"
   outputType="file"
   outputFile="output.jpg">
      <CF_MagickAction action="Geometry"
         width="120"
         percent="no"
         scaleIf="larger"/>
</CF_MagickTag>
```

```
<cimg src="pandababy.jpg" max-width="120"/>
```

**Beispiel 4.6.1:** *Beschreibung von Grafiktransformationen (ColdFusion/ ImageMagick und Roxen).*
Die Beispiellistings beschreiben jeweils die Begrenzung der Grafik `pandababy.jpg` auf maximal 120 Pixel.

*Thumbnails*, Icons, die Bildschirmanzeige im Text oder Bildausschnitte können dann ähnlich wie HTML-Ansichten von XML-Dokumenten vom CMS generiert werden.

Eine besondere Rolle kommt in diesem Zusammenhang der Verarbeitbarkeit von SVG (*Scalable Vector Graphics*) zu. Da es sich bei SVG um eine XML-Sprache zur Beschreibung von Vektorgrafiken handelt, ist eine XSL-Transformation oder zweistufige Generierung (vgl. Kapitel 2) von SVG-Grafiken analog zur Generierung Webseiten naheliegend. Um die sich daraus ergebenden Möglichkeiten voll ausnutzen zu können, sollte ein CMS insbesondere die Transformation von SVG in gängige Pixelformate leisten können, da eine integrierte Darstellbarkeit von SVG in Browsern noch nicht allgemein gegeben ist.

Dass sich zur Zeit noch keine Standardsprachen zur Beschreibung von Bild-Transformationen etabliert haben, ist im Hinblick auf Kompatibilität und Interoperabilität wenig problematisch, da die oben genannten, hauptsächlich benötigten Transformationen keine komplexe Syntax benötigen und verschiedene syntaktische Varianten somit mit relativ wenig Aufwand ineinander überführbar sein sollten (siehe Beispiel 4.6.1).

### 4.6.3 Automatische Grafikgenerierung

Neben der automatischen Transformation kann insbesondere auch die *automatische Generierung* von Grafiken helfen, Redundanzen zu vermeiden und die Präsentation von Inhalten stark vereinfachen.

#### 4.6.3.1 Textgrafiken

Automatisch generierte Textgrafiken können z. B. sinnvoll sein, um Textelemente zu erzeugen bei denen die strikte Einhaltung eines intendierten Layouts – aus Platzgründen oder zur Umsetzung von *Corporate-Design*-Vorgaben – notwendig ist. Während auf der einen Seite reines HTML und CSS kein bestimmtes Rendering garantieren können, ist auf der anderen Seite die manuelle Erstellung und Wartung von Textgrafiken mit Hilfe von Grafikprogrammen extrem aufwendig und inhaltlich nicht mit anderen CMS-Inhalten synchronisierbar. Wenn ein CMS dagegen Grafiken anhand von z. B. durch XPath-Ausdrücke berechenbare oder statische Zeichenketten und typografischen Parametern (Schriftart, Größe, Farbe, ...) generieren kann, ist sowohl grafische Integrität als auch inhaltliche Synchronisation und damit leichte Wartbarkeit gewährleistbar.

#### 4.6.3.2 Diagramme

Ähnliche Vorteile wie durch die Textgrafik-Generierung können durch eine automatisierte Generierung von Diagrammgrafiken erzielt werden. Wenn die Scriptsprache eines CMS z. B. Befehle zur Verfügung stellt, durch die im CMS vorhandene oder externe Daten geeignet visualisiert werden können, entfällt nicht nur der hohe Aufwand einer manuellen Erstellung. Insbesondere entfällt auch der für die Synchronisierung von veränderlichen Daten und Präsentationsgrafik erforderliche Wartungsaufwand. Besonders interessant ist in diesem Zusammenhang wiederum die dynamische Generierung von Diagrammen im SVG-Format.

## 4.7 Versionsmanagement

### 4.7.1 Konfligierende synchrone Versionen

Wenn nicht ausgeschlossen werden kann, dass mehrere Personen – Autoren, Redakteure, Designer, Programmierer, usw. – gleichzeitig an einem Webauftritt arbeiten, muss ein CMS Mechanismen bereitstellen, um etwaige Konflikte, z. B. bei gleichzeitiger Bearbeitung einer Datei, auszuschließen oder aufzulösen. Diese Fähigkeit ist bei CMS im Gegensatz zu herkömmlichen Webservern besonders wichtig, da einerseits durch den Einsatz eines CMS Änderungen nicht mehr – wie sonst häufig der Fall – nur durch einen einzelnen Webmaster vorgenommen werden und andererseits durch Möglichkeiten der Redundanzvermeidung die Informationskonzentration zunimmt, so dass sich insgesamt die Wahrscheinlichkeit solcher Konflikte deutlich erhöht.

Ein Weg der Konfliktvermeidung, der häufig von CMS begangen wird, bei denen sich restriktive *Workflows* definieren lassen, ist Dateien, die von einer Person bearbeitet werden, für andere zu *sperren* oder jeweils nur der im Workflow vorgegebenen Person überhaupt Schreibzugriffe zu erlauben. Dieser Weg bietet allerdings keine allgemein zufriedenstellende Lösung, da sich Workflows nicht immer so restringiert definieren lassen und häufig gerade Teamarbeiten an einem Projekt gewünscht sind. Darüber hinaus ist auch das Risiko, dass Dateien versehentlich gesperrt bleiben und nicht mehr zu bearbeiten sind, zu groß.

### 4.7.2 Diachrone Versionen und Archivierung

Neben der Unterstützung synchron existierender Dateiversionen sollten CMS auch in der Lage sein, diachrone Versionen verwalten zu können. Das heißt, dass sowohl alte Versionen einzelner Dateien als auch komplette Zustände eines Webauftritts wiederherstellbar sein sollten. Außerdem sollte nicht nur der CMS-Anwender alte Versionen einzeln einsehen können, sondern diese ggf. auch *selektiv* für die Öffentlichkeit freigeben können. Stylesheets können so – vorausgesetzt, sie haben Zugriff auf diese Informationen – entsprechend verlinkte Verweise auf ältere bzw. neuere freigegebene Versionen eines Do-

kuments generieren und damit effektiv ein teil-öffentliches Ressourcenarchiv realisieren (siehe dazu auch Abschnitt 4.9.2). Ein weiteres, unter Umständen interessantes Feature, das auf Grundlage einer kompletten Versionsverwaltung aller Ressourcen realisierbar sein sollte, ist die Ansicht des Zustands einer Site zu einem bestimmten Stichtag.

Außer zu Archivierungszwecken ist eine Versionsverwaltung, aber insbesondere auch zum Revidieren etwaiger Fehler einsetzbar. Die Notwendigkeit dieses Features ist – ähnlich wie die eines Konfliktauflösungsmechanismus – im besonderen Maße innerhalb eines CMS gegeben, da viele Informationen an mehreren Stellen durch unterschiedliche Personen verwendet werden und deshalb ungewünschte Seiteneffekte schwer vorhersehbar sein können oder erst nach einiger Zeit erkannt werden. Um entdeckte Fehler schnell rückgängig machen zu können ist es außerdem sinnvoll, wenn in den Historien von Dateien Bemerkungen zu den einzelnen Veränderungen gemacht werden können, so dass später die richtige Version einer Datei wiedergefunden werden kann.

### 4.7.3 Trennung von Entwicklungs- und Produktionssystem

Das dritte wichtige Feature im Zusammenhang mit Versionsmanagement ist, dass Veränderungen zunächst nur für den Autor selbst oder eine definierte Gruppe sichtbar sein sollten, nicht aber für die Öffentlichkeit. Genauer gesagt sollten Produktions- und Entwicklungssystem so voneinander getrennt sein, dass Veränderungen zunächst vom Autor (und evtl. anderen am Workflow beteiligten Personen) in einem möglichst exakten Abbild des Produktionssystems getestet werden können, bevor sie schließlich nach Abschluss der Tests für das Produktionssystem – also öffentliche Sichtbarkeit – freigegeben werden. Dieses Ziel ist kaum durch einfache lokale Backups erreichbar, da innerhalb eines CMS viele Abhängigkeiten zwischen zu verändernden und anderen Dateien zu erwarten sind, so dass im Extremfall – z. B. wenn Änderungen an zentralen Konfigurationen oder globalen Stylesheets vorgenommen werden sollen – ein großer Teil der CMS-Inhalte kopiert werden müssten. Darüber hinaus müsste dann bei jedem Redakteur eine Instanz des CMS in Betrieb sein.

### 4.7.4 Das Concurrent Version System (CVS)

Ein System, auf dessen Grundlage sich die drei oben genannten Hauptforderungen an eine Versionsverwaltung erfüllen lassen, ist das *Concurrent Version System* (CVS) [Pur00]. Das CVS wird im Bereich der Softwareentwicklung bereits seit Anfang der 90er Jahre verwendet und ist insbesondere bei größeren Projekten mit mehreren beteiligten Programmierern zum Standard geworden.

Kern eines CVS ist ein Lager – das sogenannte *Repository* – in dem alle diachronen Versionen aller Dateien *differentiell* und mit Metainformationen (z. B. Versionsnummer, Änderungsdatum, Änderungsbeschreibungen, ...) gespeichert sind. Das Repository liegt dabei zwar in einem gewöhnlichen Dateisystem, das *Check in* und *Check out* von Dateien erfolgt jedoch über eine CVS-eigene Schnittstelle. Diese sorgt unter anderem dafür, dass durch zeitlich überlappende Bearbeitung entstandene Konflikte beim Zurückschreiben von Dateien so aufgelöst werden, dass alle gemachten Änderungen berücksichtigt werden – es sei denn, die konkurrierenden Versionen wurden unterschiedlich an identischen Stellen verändert. In letzterem Fall müssen Konflikte manuell aufgelöst werden. Darüber hinaus erlaubt die Schnittstelle natürlich Zugriffe auf Veränderungsgeschichten und alte Dateiversionen.

Mit Hilfe eines CVS lassen sich also sowohl synchrone als auch diachrone Versionen entsprechend der aufgestellten Kriterien verwalten. Um für Benutzer vom Produktionssystem getrennte Testumgebungen zu schaffen, sind unterschiedliche Ansätze denkbar. In der Softwareentwicklung ist diese Trennung in der Regel dreifach: Außer dem zentralen Repository existiert bei jedem Entwickler eine lokale, mehr oder weniger aktueller Kopie des Quelltextbaums. Darüber hinaus können beliebige im Repository vorhandene Versionen durch Markierungen zu unterschiedlichen öffentlichen *Releases* zusammengefasst werden. Während letztere Trennung bei Software zur Zusammenstellung *stabiler* Versionen durchaus zweckmäßig ist, erscheint im Webbereich aufgrund schwächerer Abhängigkeiten und höheren Aktualitätsanforderungen eine einfache Entwicklungs-/ Öffentlichkeits-Trennung zwischen den lokalen Abzügen der Entwickler und den jeweils aktuellsten Versionen im Repository ausreichend. Um dabei aber – wie oben gefordert – ein

Testen von Veränderungen im Kontext zu erlauben, ohne gleichzeitig Kopien des Repositories zu erzwingen, muss die Standard-CVS-Funktionalität durch das CMS erweitert werden. Das CMS sollte die lokalen Bereiche der Entwickler/ Benutzer so mitverwalten, dass dort nur sich vom aktuellen Repository-Pendant unterscheidende Versionen gespeichert sind. Während einer *Test-Session* sollten dann standardmäßig vom CMS – wenn vorhanden – lokale Objekte des Benutzers und sonst die aktuelle Repository-Versionen ausgeliefert werden. Analog dazu sollte auch das Dateitransfer-Interface Dateien beim Herunterladen ausliefern und hochgeladene Dateien zunächst im lokalen Bereich speichern, bis diese durch ein *Check in*, bzw. ein sogenanntes *Commit*, bestätigt und damit ins Repository überführt werden.

## 4.8 Management von Sprachvarianten

In engem Zusammenhang zum Versionsmanagement steht die Verwaltung unterschiedlicher *Sprachvarianten* von Ressourcen. Ein CMS sollte dabei einerseits die Auslieferung passender Sprachversionen an den Besucher einer Site und andererseits den Autor bei ihrer Erstellung und Pflege so weit wie möglich unterstützen.

### 4.8.1 Aus Besuchersicht

Auf der Besucherseite sollten möglichst zwei Optionen der Sprachselektion zur Verfügung stehen: Standardmäßig sollte die Selektion automatisch anhand der in der im Browser eingestellten und über das HTTP-Anfragefeld `Accept-Language` [siehe FGM+99] an das CMS übermittelten Präferenzliste erfolgen. Zusätzlich sollte der Besucher aber die Möglichkeit haben, explizit eine bestimmte Sprache auszuwählen, um z. B. im Falle einer wenig aktuellen Übersetzung auf das Original zurückgreifen zu können. Damit der Besucher seine Entscheidung, solange er sich auf der Site befindet, nur einmal treffen muss und diese als solche für ihn transparent bleibt, empfiehlt es sich, diese durch einen sogenannten *Prestate* am Anfang des aktuellen Pfads zu kodieren. Wenn das CMS eine solche Kodierung interpretieren kann, bleibt der Besucher bei der Verfolgung relativer Links – für ihn nachvollziehbar – in

seiner gewählten Sprache und ist außerdem in der Lage, bestimmte Sprachvarianten einer Seite zu bookmarken.

### 4.8.2 Aus Autorensicht

Zur Unterstützung von Benutzern bzw. Autoren bei der Verwaltung von Sprachvarianten, muss ein CMS zuerst Konventionen festlegen, wie verschiedene Varianten voneinander getrennt und gleichzeitig als einander zugehörig markiert werden. Das Vereinen unterschiedlicher Sprachen in jeweils einer XML-Datei hat den Nachteil, dass es nicht allgemein auf alle Dateien, sondern nur auf Textdokumente angewendet werden kann, die später sprachsensitiv transformiert werden können. Außerdem können keine Standard-XML-Schemata, DTDs und XSLT-Stylesheets verwendet werden, die keine solche Sprachtrennung unterstützen. Mit den beim Apache-Webserver[5] verwendeten sogenannten *Type-Maps* – spezielle Dateien in denen eine Abbildung verschiedener HTTP-Anfragewerte auf Varianten auszuliefernder Ressourcen festgelegt werden – können zwar sehr feine und ausreichend flexible Einstellungen getroffen werden, jedoch sollte ein CMS eben genau das Treffen solcher Einstellungen so weit wie möglich automatisieren.

Das einfachste Verfahren auf dessen Basis dies möglich ist, ist Übersetzungen von Originalen konsistent durch Infix-Sprach-Tags (nach RFC1766 [Alv95]) in Dateinamen (z. B. `index.en.xml`) zu kodieren. Wenn Sprachcodes außerdem in den Metadaten angegeben sind, kann ein CMS prinzipiell selbstständig die für den Besucher geeignete Sprachvariante auswählen. Um bei der Auswahl auch die Aktualität und die Qualität einer Übersetzung berücksichtigen und den Besucher ggf. über eine nicht aktuelle Übersetzung zu informieren zu können, sollten Metadaten auch hierzu entsprechende, durch das CMS interpretierbare Informationen enthalten. Unter Umständen kann es auch sinnvoll sein, existierende Sprachvarianten in Workflows einzubeziehen, so dass versehentlich nicht aktualisierte Übersetzungen, bzw. nicht als solche markierte, ausgeschlossen werden können [siehe Zsc00].

[5] `http://www.apache.org`

## 4.9 Link- und URI-Management

Eines großes Ärgernis im *World Wide Web* sind ins Leere oder zum falschen Ziel führende Links und Bookmarks. Im Folgenden sollen Techniken diskutiert werden, auf deren Grundlage Content-Management-Systeme helfen können, dieses Problem zu vermeiden.

Ein Ansatz, der in einigen CMS Anwendung findet, ist eine interne Linkverwaltung, die z. B. durch unveränderliche Ressourcenidentifikatoren (ähnlich den in Abschnitt 1.1.3 vorgestellten URNs – dafür sorgt, dass CMS-interne Links konsistent bleiben oder zumindest Inkonsistenzen nicht unbemerkt entstehen. Dieses Verfahren *allein* löst allerdings nur einen kleinen Teil des Problems, da sowohl von außen kommende Links als auch von Besuchern gesetzte Lesezeichen auf diese Art und Weise nicht behandelt werden können.

Die einzige Möglichkeit, ins Leere führende Links grundsätzlich zu vermeiden, ist gemäß dem Motto »*Cool URIs don't change*« [BL98a] bestehende Ressourcen nicht zu löschen und bestehende URIs nicht mehr zu verändern. Deshalb sollte die Unterstützung eines CMS vor allem an dieser Stelle ansetzen.

### 4.9.1 Fehlerhafte Links durch Umstrukturierungen

Eine häufige Ursache für URI-Änderungen ist die Neu-Strukturierung von Inhalten. Da sich solche Reorganisationen nicht immer vermeiden lassen, sollten CMS Möglichkeiten schaffen, dabei die alten URIs bestehen zu lassen. Neben aus Unix-File-Systemen bekannten Dateiverknüpfungen, mit Hilfe derer sich Dateien über beliebige Pfade adressieren lassen – mit denen sie *verknüpft* sind, sind im WWW-Kontext insbesondere sogenannte *Redirects* geeignete Lösungen.[6]

Die Verwaltung solcher *Redirects* sollte, um bei der Erstellung neuer Dateien Konflikte mit ehemalig vorhandenen zu vermeiden, direkt im Repository erfolgen – idealerweise so, dass nicht nur Verweise zu einzelnen Dateien

[6] Bei einem *Redirect* schickt der Webserver, bzw. das CMS den Status-Code 301 [FGM+99] (»*Moved permanently*«) an den Klienten zurück und informiert ihn gleichzeitig über den neuen URI.

oder Ordern, sondern auch zu Ordnerinhalten möglich sind, so dass beim Verschieben eines Ordners nur ein Redirect benötigt wird.

Als zusätzliche Vereinfachung ist außerdem eine optionale automatische Einrichtung von Redirects beim Verschieben von Ordnern oder Dateien denkbar.

### 4.9.2 Fehlerhafte Links durch gelöschte Dateien

Die zweite Ursache für fehlerhafte Links ist das Löschen von Dateien, wenn diese *scheinbar* nicht mehr benötigt werden oder ihre Inhalte nicht mehr aktuell sind. Ein solches Löschen können CMS vollständig überflüssig machen. Durch Setzen von Metainformationen zum aktuellen Status und zur Gültigkeitsspanne, können Dateien von Stylesheets so aufbereitet werden, dass die Ressource über ihren URI permanent adressierbar bleibt, der Besucher aber ggf. über ihre abgelaufene Gültigkeit und ihren aktuellen Status in Kenntnis gesetzt wird. Besonders hilfreich ist dabei, wenn das CMS die Metadaten soweit selbständig auswerten kann, dass der Autor selbst automatisch – z. B. per E-Mail – von einem Gültigkeitsablauf informiert wird.

Weitere in diesem Zusammenhang sinnvolle Metadaten sind mehr oder weniger qualifizierte verlinkte Verweise auf Nachfolger (z. B. »*Ersetzt durch die Prüfungsordnung vom 1.4.2000.*«) oder die gerade aktuelle Versionen.

## 4.10 Workflow-Management

In den Anfängen des WWW war es üblich, dass fertige Texte nach Abschluss aller redaktionellen Prozesse, evtl. sogar nach ihrem Druck, wenn sie für ausreichend wichtig eingestuft waren, an einen Webmaster weitergegeben wurden. Dieser war dann dafür zuständig, die Dokumente unter Umständen nach HTML zu konvertieren, die Form für das Web aufzubereiten und sie über einen Webserver zu publizieren.

Diese Art des Vorgehens ist heute nicht mehr praktikabel. Auf der einen Seite werden nicht mehr nur ausgewählte, sondern *alle* für die Öffentlichkeit bestimmten verfügbaren Informationen im Internetangebot eines Unternehmens oder einer Organisation erwartet. Zum anderen wird eine dem Medi-

um entsprechende Aktualität erwartet. Um beide Erwartungen zu erfüllen, müssten in etwa ebenso viele Webmaster wie Autoren beschäftigt werden.

Das heißt, dass Zeitschriftenredakteure, PR-Beauftragte in Firmen oder Pressestellenmitarbeiter von Organisationen und Universitäten selbst im Netz publizieren können müssen. Eine der daraus resultierenden Anforderungen an ein CMS ist die Fähigkeit, Inhalte nicht nur als Ergebnisse eines abgeschlossenen redaktionellen Prozesses behandeln zu können, sondern sie in ihren verschiedenen Entwicklungsstufen repräsentieren zu können. Es liegt also nahe, die gewohnten redaktionellen Prozesse in ein Content Management System einzubeziehen und diese so weit wie möglich zu unterstützen und zu automatisieren.

Der typische redaktionelle Zyklus, der heute von vielen CMS unterstützt wird sieht in der Regel vier Phasen vor: Zunächst werden digitale Inhalte, sogenannte *Assets*, von Autoren oder Grafikern erstellt. In der zweiten Phase werden diese Assets von dafür verantwortlichen Redakteuren auf ihre inhaltliche, formale und gestalterische Korrektheit überprüft und entweder an die Autoren, evtl. mit Kommentaren versehen, zurückgegeben oder in die Publikationsphase des Zyklus weitergereicht. Erst in der Publikationsphase erfolgt der Schritt, der bei der Arbeit mit herkömmlichen Webservern ganz am Anfang steht: die freigegebenen Inhalte werden im Internet (oder auch im Intranet) veröffentlicht. In der letzten Phase muss dafür gesorgt werden, dass die publizierten Inhalte archiviert werden und auch in Zukunft verfügbar und auffindbar sind.

## 4.11 Benutzerverwaltung und Rechtevergabe

Um auch im Bereich der Verwaltung von Benutzern unnötige Redundanzen zu vermeiden, muss ein CMS neben einer eigenen Benutzerverwaltung über Möglichkeiten zur Anbindung an externe Server verfügen. Wie bereits in Abschnitt 4.2.4 dargestellt, sollte dazu das *Lightweight Directory Access Protocol* (LDAP) clientseitig unterstützt werden. Zusätzlich kann außerdem eine Unterstützung von Standard-Unix-Passwort-Dateien, insbesondere in Umgebungen ohne zentralen LDAP-Server oder auch zur Vermeidung von Sys-

temabhängigkeiten sinnvoll sein.

Besonders hohe Anforderungen sind an die Rechteverwaltung eines CMS gestellt. Für jeden gegen die Benutzerverwaltung authentisierten Benutzer und die Klasse nicht-authentisierter Benutzer müssen Zugriffsrechte zum einen nach ihrer Art und zum anderen nach ihrem Ort (Datei, Verzeichnis, oder Verzeichnisbaum) modifizierbar sein. Die Zugriffsart sollte dabei neben gewöhnlichem Schreib- bzw. Leserecht auch in Bezug auf die Ausführung von Scripts, bzw. die Ausführung bestimmter Scriptbefehle und in Bezug auf die Veränderbarkeit von Rechten selbst (und deren Delegierbarkeit) differenziert sein. Ein Ansatz, die Komplexität der Rechtevergabe in dem durch *Benutzer* $\times$ *Art des Rechts* $\times$ *Ort* aufgespannten Raum möglichst einfach zu halten, ist zum einen – ähnlich wie in Unix-Dateisystemen üblich – Benutzer Gruppen zuordnen zu können und zum anderen Rechte grundsätzlich innerhalb der Dateisystemhierarchie nach unten zu vererben.

## 4.12 Hochschulspezifische Anforderungen

### 4.12.1 Strukturelle Besonderheiten an Hochschulen

Die strukturellen Voraussetzungen für eine übergreifende Einführung neuer Software in einer Hochschule heben sich deutlich von jenen an Unternehmen ab. Zwar lässt sich die Organisationsstruktur einer Hochschule auf eine Hierarchie abbilden, die Kompetenzen von übergeordneten Instanzen gegenüber untergeordneten sind, aufgrund der weitgehenden Autonomie von Lehrstuhlinhabern – über die Freiheit von Forschung und Lehre hinaus – aber nur sehr begrenzt. Hinzu kommt, dass gerade im Bereich der *Webpräsenz* sich die Kompetenzansprüche verschiedener Lehrstühle (z. B. Informatik, Wirtschaftsinformatik, Design, Pädagogik, ...) und Einrichtungen (z. B. Bibliothek, Rechenzentrum, Pressestelle) stark überschneiden und die Beziehungen zwischen den Vertretern der unterschiedlichen Parteien in der Regel nicht unvorbelastet sind. Die Konsequenz, die sich hieraus ergibt, ist, dass eine Software nur dann die Chance hat hochschulweit eingesetzt zu werden, wenn Vertreter aller Parteien nicht nur bei der Bedarfsanalyse, sondern auch an Planungs- und Auswahlprozessen beteiligt werden.

Außer durch ihre dezentrale Organisationsstruktur unterscheidet sich die Hochschule aber auch durch ihre potentiell deutlich höhere Anwenderdiversität von einem durchschnittlichen Unternehmen. Die Dimensionen dieser Diversität lassen sich z. B. in Interessengruppen (Professoren, Studenten, wissenschaftliche Mitarbeiter, nicht-wissenschaftliche Mitarbeiter) sowie Grad der Expertise und Fachrichtung klassifizieren. Durch diese von der Organisationsstruktur teilweise abweichenden Dimensionen wird sowohl die Bildung von Auswahlgremien als auch insbesondere das Finden eines tragfähigen Konsenses bezüglich der Prioritäten der geforderten Eigenschaften einer Software erschwert.

### 4.12.2 Spezielle Aspekte bei der Wahl eines CMS

Aus den im vorangegangenen Abschnitt dargelegten Gründen ist die Einführung einer neuen Software mit dem Ziel ihrer systemweiten Nutzung allgemein problematischer als bei durchschnittlichen Unternehmen. Bei der Ablösung eines zentralen, herkömmlichen Webservers durch ein zentrales Web-Content-Management-System im Speziellen, kommt erschwerend hinzu, dass sowohl die Anzahl betroffener Nutzer bzw. Autoren als auch die Menge bereits bestehender Inhalte deutlich größer ist als zum Beispiel im Falle von E-Learning-Systemen. Daher spielt das Kriterium der *Abwärtskompatibilität* (zum vorher verwendeten Webserver) bei der Auswahl CMS für eine hochschulweite Nutzung eine besondere Rolle. Nur dadurch, dass erstens gewohnte Arbeitsabläufe weiterhin unterstützt werden und zweitens kein Zwang zu einer sofortigen Neu-Einstellung von Inhalten auferlegt wird, kann letztlich eine ausreichende Akzeptanz des Systems erzielt werden.

Der von einigen CMS-Anbietern als *Chance zum Neuanfang* und zur Entledigung von Altlasten propagierte Zwang zur Neuerstellung von Inhalten ist im Umfeld der Hochschule aufgrund ihrer Organisationsstruktur, der Menge an vorhandenem Web-Content und der dem gegenüberstehenden Ressourcenknappheit ein kaum zu überwindendes Hindernis. Es sollte daher vielmehr Aufgabe des CMS und nicht die der Anwender sein, bestehende HTML-Dokumente zu einem möglichst hohen Grad in das *Corporate Look and Feel* eines neuen Webauftritts zu integrieren. Wenn ein CMS außerdem im

Hinblick auf unterstütze Arbeitsabläufe (z. B. FTP-Upload von Ressourcen) abwärtskompatibel zu einem zuvor eingesetzten Webserver ist, kann die in einer Hochschule am meisten Erfolg versprechende Strategie einer allmählichen Migration eingeschlagen werden. Allmähliche Migration soll in diesem Zusammenhang heißen, dass *nach* der Umstellung von Webserver auf CMS, das Angebot an Fähigkeiten in Bezug auf die Auszeichnung von Inhalt und die Unterstützung von Arbeitsabläufen sukzessiv, je nach Bedarf von den Benutzern ausgeschöpft werden kann.

# 5 Relaunch des Webauftritts der Universität Bielefeld

## 5.1 Der Vorherzustand

Der Webauftritt der Universität Bielefeld vor September 2001 – die Startseite ist in Abschnitt 5.1 dargestellt – war auf den ersten Blick vor allem durch sein unübersichtliches, uneinheitliches Erscheinungsbild geprägt. Das Aussehen der Seiten variierte so sehr, dass man nach Anklicken eines internen Links oft das Gefühl hatte, die Seiten der Universität bereits verlassen zu haben. Navigationsmöglichkeiten waren – von den meist überladenen Startseiten abgesehen –, wenn sie überhaupt vorhanden waren, in Form von Links über die Seiten verteilt. Um Informationen zu finden, blieb wegen der unsystematischen Strukturierung und der z.T. fehlenden Verlinkung oft nur der Weg über die interne Suchmaschine. Auch das Wiederfinden von häufiger benötigten Stellen war problematisch, da durch die Verwendung von *Framesets* viele Seiten nicht ohne Weiteres mit Lesezeichen versehen werden konnten.

Die Zuverlässigkeit und Aktualität des Inhalts der Seiten war in der Regel schwer einschätzbar und verifizierbar, da Datum der letzten Änderung, sowie Name und E-Mail-Adresse der inhaltlich und technisch Verantwortlichen entweder nicht dargestellt oder wenig vertrauenswürdig waren.

Hauptursache für den in Funktion, Struktur und Gestaltung stark heterogenen, letztlich sehr unbefriedigenden Auftritt war, dass die Benutzer des Webservers sowohl bei der Konzeption als auch bei der Implementation ihrer Seiten auf sich allein gestellt waren. Das heißt, Autoren und »Webmaster« mussten erstens bei jeder einzustellenden Seite selbst entscheiden, ob sie über die reine Präsentation des Inhalts hinausgehende Features – wie Na-


**Abbildung 5.1:** Die alte Startseite der Universität Bielefeld

vigationselemente, Metainformationen und einheitliche Gestaltung – anbieten wollen und zweitens diese ggf. auch selbst implementieren und warten. Es fehlte also sowohl an einer für Vorgaben und Koordinationsaufgaben zuständigen Anlaufstelle als auch auch an technischen Hilfsmitteln, um einen grundlegenden Qualitätsstandard garantieren zu können. Die Qualität der einzelnen Seiten der Universität und ihrer Einrichtungen konnte damit – abhängig von den jeweilig verfügbaren personellen und finanziellen Ressourcen – ohne untere Beschränkung variieren. Aufgrund fehlender Möglichkeiten zur Trennung von Inhalt und Präsentation und des dadurch bedingten hohen Implementations- und Wartungsaufwands tendierte die Qualität dabei in der Regel zum unteren Rand des Variationsspektrums.

## 5.2 Vorbereitungsphase

### 5.2.1 Das Web-Team

Um dem oben beschriebenen, unbefriedigenden Zustand des Webauftritts zu begegnen, beauftragte das Rektorat der Universität Bielefeld Anfang des Jahres 1999 das aus Vertretern verschiedener Einrichtungen bestehende Web-Team mit einer technischen und redaktionellen Neu-Konzeption. Aus der Arbeit dieses Web-Teams gingen bis Ende des selben Jahres unter anderem folgende Zwischenergebnisse hervor:

- Beim neuen Webauftritt sollten Inhalt und Layout möglichst getrennt voneinander repräsentiert sein.
- Zur Vereinfachung der Content-Verwaltung und Umsetzung der Inhalt-Layout-Trennung sollte ein noch auszuwählendes Content-Management-Systems eingesetzt werden.
- Ein Vorschlag zur Navigationsstruktur der ersten drei Ebenen des Webauftritts. Ein wesentliches Merkmal der Navigationsstruktur sollte dabei eine Aufteilung in eine einerseits an die Organisationsstruktur der Universität angelehnte Gliederung und andererseits eine benutzergruppenspezifische Gliederung sein (siehe Abbildung 5.4 auf Seite 132).

- Ein an das Corporate-Design der Universität angelehntes Pflichtenheft zum Design der Startseite und des neuen Webauftritts insgesamt (unter besonderer Berücksichtigung technischer Niedrigschwelligkeit).
- Die Empfehlung zur Ausschreibung der Gestaltung des Webauftritts (bzw. des Prototyps der ersten drei Navigationsebenen) gemäß des Pflichtenhefts an professionelle Agenturen.
- Die Empfehlung zur Einrichtung der Stelle eines hauptamtlichen Koordinators (»Informationsmanagers«), sowie die Einstellung einer zweiten Person, verantwortlich für Programmierung und Benutzersupport (siehe Abschnitt 5.2.2).
- Einbeziehung von Vertretern, bzw. Webbeauftragten aller Einrichtungen und Fakultäten bei anstehenden Entscheidungsprozessen.

### 5.2.2 Personelle Maßnahmen

Entsprechend der Empfehlung des Web-Teams wurde zum 1.9.2000 zur Leitung und Koordination des neuen Webauftritts vom Rektorat die neue, an der Pressestelle angesiedelte Stelle des Informationsmanagers geschaffen. Außerdem wurde eine für die Migrations- und Aufbauphase auf 2 Jahre befristete halbe Mitarbeiterstelle zur texttechnologischen Konzeption und zur Implementation von Stylesheets und sonstiger Software geschaffen. Vom HRZ sollte zudem ein Mitarbeiter zur Pflege von Hardware und CMS abgestellt werden.

Das Aufgabengebiet des Informationsmanagers sollte im Einzelnen folgende Punkte umfassen:

- Leitung des Web-Projektteams
- Kontinuierliche Umsetzung und Weiterentwicklung des Konzeptes der Universität Bielefeld zur Präsentation im Internet auf Basis eines verteilten und integrierten Systems
- Optimierung der Kommunikation mit den Einrichtungen bei der Informationsbeschaffung und der Pflege der WWW-Präsentation

- Konzeption und Koordination der Entwickleraktivitäten im Bereich der Web-Templates (Implementierung des Content-Management-Systems)
- Beschaffung, Aufbereitung und redaktionelle Bearbeitung von Webinhalten
- Erste prototypische Realisierung von Beispielanwendungen (Datenbanken, Internetauftritt, Navigationsstrukturen) als Diskussionsgrundlage
- Kooperation mit dem BIS-Projekt (Personalverzeichnis, Vorlesungsverzeichnis) und Unterstützung bei der Implementierung, Qualitätsmanagement (aktuelle und verlässliche Informationen)
- Schaffung eines konsistenten Erscheinungsbilds (visuelle und gestalterische Einheitlichkeit)

### 5.2.3 Auswahl eines Content-Management-Systems

Für die Auswahl des CMS für die Universität wurde vom Web-Team ein detailliertes Pflichtenheft, sowie eine Profilbeschreibung und ein Fragenkatalog entworfen. Nach einer ersten, durch studentische Hilfskräfte durchgeführte, Marktanalyse kamen folgende acht Produkte in die engere Auswahl: *ZOPE*, *Roxen Platform*, *Hyperwave*, *NPS4*, *Iplanet Application Server*, *VIP Content Manager*, *Silverstream*. Von diesen Produkten erfüllte zum damaligen Zeitpunkt lediglich *Roxen Platform* alle K.O.-Kriterien, wie die Möglichkeit zur Übernahme alter HTML-Inhalte, LDAP-Schnittstelle, delegierbare Rechtevergabe, XML/XSL-Unterstützung und einen niedrigen Preis. Alle anderen Produkte scheiterten an mehr als einem K.O.-Kriterium.

#### 5.2.3.1 Charakteristika des Roxen-CMS

Das Web-Content-Management-System der 1994 gegründeten schwedischen Firma Roxen IS[1] wurde in der Programmiersprache Pike[2] entwickelt und besteht aus dem freien Roxen-Webserver und der kommerziellen Applikation

[1] `http://www.roxen.com`
[2] `http://pike.ida.liu.se/`

*Roxen-Platform*, die die meisten CM-Fähigkeiten implementiert. Im Folgenden sollen die Eigenschaften des Roxen CMS kurz charakterisiert und mit den im vorangegangenen Kapitel aufgestellten Anforderungen verglichen werden.

Zum Zeitpunkt der Auswahl eines CMS für die Universität Bielefeld Anfang des Jahres 2000 war Roxen, wie gesagt, das einzige System, das den Mindestkriterien genügte. Vergleicht man die in Tabelle 5.1 dargestellten Charakteristika mit den im vorangegangen Kapitel dargestellten Anforderungen, stellt man fest, dass diese zu einem großen Teil erfüllt sind, jedoch insbesondere eine Unterstützung erweiterbarer und durchsuchbarer Metadaten sowie eine Möglichkeit zur Veröffentlichung von Ressourcen-Versionen (siehe Abschnitt 4.7.2) fehlt.

Aufgrund dieser fehlenden Features kann Roxen zur Zeit nur eingeschränkt als Dokumentenarchiv oder zur Verwaltung digitaler Assets (z. B. Grafiken und Sounds) genutzt werden. Die Konsequenzen daraus sind weitreichend. Da ein Dokumentenarchiv ebenso wie ein zentrales Archiv für multimediale Ressourcen benötigt wird, muss unter Umständen zusätzliche Spezialsoftware angeschafft werden. Dies bringt sowohl im Bereich der Systemwartung wie auch bei der Verwaltung und Verarbeitung von Inhalten unerwünschte Redundanzen mit sich. Hinzu kommt, dass mit jeder zusätzlichen Software auch zusätzliche Schnittstellenprobleme entstehen, da Inhalte und insbesondere auch Metadaten aus Bild- und Dokumentarchiven natürlich auch potentielle Webinhalte sind und vom CMS deshalb zugänglich und über Stylesheets für ihre Präsentation aufbereitbar sein müssen.

Leider ist auch die Möglichkeit zum Einbinden von Content anderer Systeme in Roxen nur unvollkommen realisiert. Zwar lassen sich mit Hilfe der Roxen-eigenen eingebetteten Scriptsprache RXML per HTTP-Anfrage beliebige Inhalte einbinden und auch per XSLT transformieren; dadurch, dass Aufbereitung und Auslieferung von Inhalten aber lediglich durch einen Prozess abgewickelt werden, ist das gesamte System während des Wartens auf einzubindende Ressourcen blockiert. So ist eine Syndizierung von Content ohne Gefährdung der Operabilität des Systems nur über den Umweg von Datenbankpuffern realisierbar (siehe Abschnitt 5.6.3).

| | | |
|---|---|---|
| **unterstützte Protokolle** | | |
| HTTP | ⊕ | allerdings nur HTTP1.0, keine WebDAV-Erweiterungen |
| FTP | ⊕ | Löschen von Ordnern und Dateien sowie Zugriff auf Metadaten jedoch nicht möglich |
| LDAP | ⊕ | |
| **Aufbereitung von Ressourcen: Stylesheets/ Scripting/ Transformationen** | | |
| XSLT | ⊕ | eigener XSLT-Prozessor (ohne echte Namensraumunterstützung) |
| einbettbare Scriptsprachen | ⊕ | eigene Scriptsprache RXML mit XML-konformer Syntax (aber auch andere Sprachen wie Perl oder PHP prinzipiell möglich) |
| Zugriff auf dokumentexterne Quellen | ⊕ | auf Verzeichnisinformationen und Metadaten über XPath-Erweiterungs-Funktion; auf Datenbanken, LDAP- und HTTP-Server über RXML – *Anfragen auf CMS-externe Ressourcen blockieren allerdings das System* |
| Grafik-Transformationen | ⊕ | sowohl per Scriptsprache als auch durch spezielle Metadaten; allerdings keine SVG-Unterstützung |
| *FO-Fromatter* | ⊖ | |
| Caching | ⊕ | explizites Caching Teilbaum-Caching von XSL-Transformationen, ähnlich wie in Abschnitt 4.4.2 beschrieben |
| **Metadaten** | | |
| frei definierbare Metadaten | ⊖ | vorgegebener Satz an »*live Properties*« (Titel, Schlüsselwörter, Beschreibung, Autor, Gültigkeitsspanne) |
| WebDAV-Unterstützung | ⊖ | weder kompatibles Format noch Schnittstelle |
| durchsuchbar | ⊖ | |
| **Management von Ressourcen-Varianten und Versionen** | | |
| CVS-Dateisystem | ⊕ | |
| Entwicklungs-»*Sandbox*« | ⊕ | Privater »Edit-Bereich« für jeden Benutzer |
| Archiv-Funktionalitäten | ⊖ | Nur authentisierte Benutzer können auf alte Ressourcen-Versionen zugreifen. |
| Sprachvarianten-Unterstützung | ⊕ | für alle Arten von Ressourcen (auch XSL-Stylesheets) |
| **Verwaltung von Benutzern und deren Rechten** | | |
| Rechtevergabe | ⊕ | flexibel und delegierbar, jedoch keine Einschränkung von Ausführungsrechten möglich |

**Tabelle 5.1:** Eigenschaften des Content-Management-Systems Roxen Platform

## 5.3 Migration und Umstellung auf das CMS

### 5.3.1 Phasen

Die Migration des Webauftritts der Universität Bielefeld vom herkömmlichen Webserver auf das Roxen-CMS kann grob in drei Phasen unterteilt werden:

1. *Entwicklungsphase (9/2000-9/2001):*
   a) Eins-zu-Eins-Übertragung des kompletten Inhalts des Webservers auf das CMS, einschließlich Zugriffsrechten, Verzeichnisstruktur und Metadaten
   b) Entwicklung von Stylesheets und zentralen Universitätsseiten (mit an die alten Inhalte angepasster Verlinkung)
   c) Pilotprojekte einzelner Einrichtungen und Fakultäten
   d) Schulung zukünftiger Benutzer (mit an die alten Inhalte angepasster Verlinkung)
   e) *Generalprobe der Umstellung (8/2001):* Erneute Übertragung der Webserverinhalte (auch zur Abschätzung des benötigten Zeitaufwands) mit Anschließenden Tests auf Inhalt, Darstellung und Verlinkung der Seiten und CMS-Performanztests unter Belastung
2. *Umstellung (8.9.2001)*
   a) Sperren der Schreibzugriffe auf Webserver
   b) Erneute Übertragung von Webserverinhalten, Zugriffsrechten und Metadaten
   c) Umstellung des Nameservereintrags von Webserver auf CMS
3. *Fortsetzung der Migration (9/2001 – )*
   - Weitere Schulungen
   - Allmähliche Ausnutzung der CMS-Features durch weitere Einrichtungen und Fakultäten
   - Weiter- und Neu-Entwicklung von Stylesheets

- Migration von Fakultäten und Einrichtungen mit ursprünglich eigenen Webservern

Mit der Umstellung auf das CMS im September 2001 war also die Migration nicht abgeschlossen. Nach außen erkennbar hatten sich nur die zentralen Seiten der Universität, bzw. die obersten drei Ebenen der neuen Navigationshierarchie (siehe Abbildung 5.2 auf Seite 129) sowie die Seiten einiger an Pilotversuchen beteiligter Einrichtungen verändert. Der Webauftritt der anderen, vorher auf dem zentralen Webserver vertretenen Einrichtungen hatte sich nach außen hin nicht verändert. Die auffälligste Neuerung, mit der auch die Benutzer und Autoren konfrontiert waren, die zunächst nicht von den neuen Features des CMS profitieren wollten oder konnten, war, dass aufgrund des CVS-Systems[3] Änderungen an Dateien zunächst freigegeben werden mussten, um sie nach außen sichtbar zu machen.

### 5.3.2 Migration von Fakultäten und anderen Einrichtungen

Die der Migration zugrundeliegende Strategie war also, wie in Abschnitt 4.12.2 bereits dargelegt, die Benutzer an den verschiedenen Einrichtungen durch die Umstellung nicht zur Änderung ihrer Arbeitsabläufe zu zwingen, sondern gewohnte Arbeitsabläufe weiterhin zu unterstützen und zusätzlich das Angebot zu machen, die durch das CMS geschaffenen, neuen Möglichkeiten jederzeit testen und nach und nach immer vollständiger nutzen zu können. Der typische Ablauf einer solchen allmählichen Ausnutzung der CMS-Features durch eine Einrichtung ist dabei folgender:

1. Optional: Teilnahme an einer Schulung zur Bedienung des CMS
2. Persönliches Gespräche mit Informationsmanager und/oder Mitarbeitern
   - Klärung von Bedürfnissen und Interessen
   - Demonstration und Erläuterung von CMS-Features
   - Evtl. probeweise Anwendung des Standard-Stylesheets auf Seiten der Einrichtung

[3] siehe Abschnitt 4.7.4

- Bei bereits vorhandenen Layoutvorstellungen oder Layoutvorlagen ggf. Implementation einer ersten Arbeitsversion einer Spezialisierung des Standard-Stylesheets
- Absprache der weiteren Vorgehensweise

3. ggf. Neustrukturierung und Optimierung der alten Inhalte
4. ggf. Entwicklung der Stylesheetspezialisierung durch Vertreter der Einrichtung mit Hilfe von Informationsmanager oder Mitarbeitern.
5. Freischaltung des neuen Auftritts
6. ggf. Entwicklung spezialisierter XML-Dokumentgrammatiken, Weiterentwicklung der Stylesheetspezialisierung und Ausweitung der Nutzung von CMS-Features wie Sprachvarianten, Grafikgenerierung und Datenbankanbindungen.

### 5.3.3 Übernahme von HTML-Inhalten und Metadaten

Voraussetzung für die Umsetzung der Strategie einer freiwilligen, allmählichen Migration nach der Umstellung auf das CMS war, dass HTML-Inhalte des alten Webservers in das neue CMS übernommen werden konnten. Dies sollte – entsprechend der Strategie – möglichst wenig invasiv, das heißt ohne Veränderungen an den einzelnen HTML-Dokumenten selbst geschehen. Trotzdem sollte es aber möglich sein auch die alten Inhalte, auf Wunsch, in der neuen Gestaltung darstellen zu können.

Obwohl die Aufbereitung von Seiten auf Grundlage von XSL-Transformationen erfolgen sollte und diese eigentlich nur auf wohlgeformten XML-Dokumenten nicht aber auf SGML-Anwendungen wie HTML definiert sind,[4] konnte diese Strategie vollständig umgesetzt werden. Ermöglicht wurde dies durch ein spezielles Roxen-Modul, das, anhand von Heuristiken, auch fehlerhaftes HTML zufriedenstellend für XSL-Transformationen *on-the-fly* in wohlgeformtes XML transformiert. Um alte HTML-Dateien im neuen Layout dar-

[4] Im Gegensatz zu XML-DTDs erlaubt die SGML-DTD von HTML das Auslassen bestimmter, inferierbarer Markierungen. Z. B. erlaubt sie die Verwendung leerer Tags oder das Auslassen von Endtags.

zustellen zu können, mussten so lediglich die jeweiligen, für das Default-Stylesheet verantwortlichen Metadaten über Roxens Webinterface oder mittels eines Scripts verändert werden. Alle weiteren Korrekturen, wie z. B. das Entfernen von visuellen Auszeichnungen (Farbattributen, Fontelementen, usw.) oder bestimmten überflüssigen Elementen, konnten so zunächst ohne weitere manuelle Eingriffe von Stylesheets bzw. vom XSLT-Prozessor vorgenommen werden.

Problematisch erschien zunächst die Extraktion der Metadaten aus den HTML-Dateien (z. B. Titel, Schlüsselwörter, Beschreibung, ...) und Einspeisung dieser in die Roxen-eigene Metadatenverwaltung. Da das Roxen-CMS über keine Standardschnittstelle zur Metadatenmodifikation verfügt (weder über WebDAV noch über einen FTP-Umweg) wurden die Metadaten schließlich – auch aus Geschwindigkeitsgründen – durch ein Batch-Script direkt in dem CVS zugrundeliegenden Dateisystem eingetragen. Bei der Migration einzelner Einrichtungen von eigenen Webservern wurden später Metadaten durch kombinierte XSLT/RXML-Scripts aus den jeweiligen HTML-Dokumenten in die Roxen-eigene Verwaltung überführt.

### 5.3.4 Aufgetretene Probleme

#### 5.3.4.1 Benutzerverwaltung und Verzeichnisrechte

Der Hauptgrund für die Verzögerung der Umstellung auf das CMS bis September 2001 – geplant war April 2001 – waren Probleme mit der Verzeichnisrechtverwaltung, bzw. deren Anbindung an die zentrale Benutzerverwaltung im Hochschul-Rechenzentrum. Zum einen war die manuelle Korrektur von Rechten sehr vieler Benutzer in sehr vielen Verzeichnissen über das Roxen-eigene Webinterface wegen der extrem großen resultierenden Matrizen (mehr als 10000 Einträge) umständlich und zeitaufwendig. Zum anderen sollten Rechte delegierbar sein, so dass der Hauptadministrator seine Rechte für bestimmte Unterverzeichnisse, z. B. an Webbeauftragte von Fakultäten weitergeben kann und diese ihrerseits wiederum Rechte weiterverteilen, aber auch wieder entziehen können. Eine solche dezentrale Rechtevergabe war Über Roxens Webinterface schwer zu realisieren. Da es zusätzliche Probleme

mit einer LDAP-Anbindung gab, wurde nach vielen Verhandlungen mit der Herstellerfirma *Roxen IS* schließlich eine andere Lösung gewählt:

Auf eine LDAP-Verbindung zwischen zentraler Benutzerverwaltung und CMS bzw. dessen Verzeichnisrecht-Verwaltung wurde verzichtet. Stattdessen werden alle Verzeichnisrechte über die zentrale Benutzerverwaltung, bzw. deren Webinterface geregelt. Diese exportiert bei Änderungen Unix-konforme Gruppen- und Passwortdateien, die vom CMS im laufenden Betrieb importiert werden. So konnte eine exakt den Anforderungen entsprechende Funktionalität erzielt werden. Durch die Verortung von Verzeichnisrechten in der zentralen Benutzerverwaltung wurde außerdem eine – in diesem Punkt sonst schwer zu realisierende – Unabhängigkeit vom verwendeten CMS erreicht und durch die Exportierbarkeit der Daten eine interoperable Schnittstelle geschaffen.

#### 5.3.4.2 Performanz

Nach der Umstellung des Webservers auf das CMS im September 2001 gab es zunächst erhebliche Performanzprobleme. Hauptursache dafür war, dass unerwarteterweise eine alte Version des CMS eingesetzt werden musste, die noch über keine Möglichkeiten zum Caching von XSLT-Transformationsergebnissen verfügte. Dass das aktuelle *Release* nicht verwendbar war, stellte sich nicht bei den Belastungstests heraus, sondern erst nach dem Import der Verzeichnisrechte – also zu einem Zeitpunkt als eine weitere Verschiebung des Umstellungstermins nicht mehr möglich war.

Weitere Gründe für die zunächst schlechte Performanz waren, neben den fehlenden Caching-Möglichkeiten:

- Generierung Sprach- und browserspezifischer HTML-Ausgaben, so dass zum damaligen Zeitpunkt Proxy-Cache-Software aufgrund fehlender *Vary-Header*-Unterstützung (vgl. Abschnitt 4.2.1), nicht verwendet werden konnte.

- unerwartet hohe, durch vorherige Tests nicht vorhersagbare Last durch viele konkurrierende Operationen gleichzeitig eingeloggter Autoren

- aufgrund des neuen Webangebots nahezu verfünffachte Hit-Rate: Maximal etwa 500.000 Anfragen pro Tag gegenüber maximal etwa 100.000 in der Planungsphase
- für die gesteigerte Hit-Rate inadäquate Hardwareausstattung des Servers
- Verwendung nur eines der beiden Prozessoren durch das CMS

Durch die Umstellung auf browserunspezifische HTML-Aufbereitung und die nachfolgende Verwendung von Proxy Caches[5] sowie Optimierungen in CMS-Einstellungen und Stylesheets und den Einsatz schnellerer Hardware konnten die Performanzprobleme bis April 2002 vollständig behoben werden.

## 5.4 Der neue Webauftritt

### 5.4.1 Gestaltung

Grundlage der Gestaltung des neuen Webauftritts der Universität Bielefeld bildete ein Entwurf der Firma Cyberpark[6], der bereits in der Vorbereitungsphase (siehe Abschnitt 5.2) ausgewählt und in Auftrag gegeben wurde. Die Gestaltung der Startseite (Abbildung 5.3 auf Seite 130) wurde abgesehen von wenigen Korrekturen vollständig übernommen. Die Gestaltung der Folgeseiten, die zunächst nur für die obersten zwei Ebenen der Navigationshierarchie konzipiert war, musste dagegen in einigen Punkten überarbeitet werden. Hauptziel der Änderungen war, im Einklang mit der Spezifikation des Corporate Design der Universität[7] ein *tragfähiges*, *flexibles* und zurückhalten-

[5] Zuerst wurde die Webseite in die standardmäßig innerhalb der Universität verwendeten Proxy Caches einbezogen. Später wurde zusätzlich ein inverser Proxy Cache (siehe Abschnitt 4.4.2.1) vor das CMS geschaltet.

[6] `http://www.cyberpark.de`

[7] Im Auftrag des Rektorats hat das *Audiovisuelle Zentrum*, eine Serviceeinrichtung der Universität, in deren Aufgabengebiet u.a. grafische Gestaltungsarbeiten liegen, 1997 eine genaue, insbesondere für Print-Sachen vorgesehene Spezifikation des *Corporate Design* hochschulintern in Form eines Handbuchs veröffentlicht. Für das Weblayout waren insbesondere Angaben zum Aussehen und Variationsmöglichkeiten des Universitätslogos, sowie Angaben zu Farben und zu verwendenden Zeichensätzen maßgeblich.

des Gerüst für *alle* offiziellen Seiten der Universität und ihrer Einrichtungen und Fakultäten zu schaffen. Dabei musste insbesondere folgenden besonderen Bedingungen Rechnung getragen werden:

- sehr breite und tiefe Navigationshierarchie bedingt durch die extrem große Menge an Seiten
- große Längenheterogenität der Navigationsmenü-Einträge durch zum größten Teil automatisch aus Metadaten gewonnene, also nicht zweckspezifisch konzipierte, Titel
- möglichst niedrige technische Anforderungen an den Browser des Benutzers (alle Seiten sollten prinzipiell auch mit sehr alten Browsern und auch mit reinen Textbrowsern benutzbar sein)
- Spielraum für die einzelnen Einrichtungen und Fakultäten das Standardlayout ihren Bedürfnissen entsprechend zu modifizieren ohne dabei den Wiedererkennungseffekt zu verlieren

Aus diesen speziellen Anforderungen ergab sich die Notwendigkeit eines Layouts, das in einigen Punkten von sonst üblichen Standard-Weblayouts abweicht. Aufgrund der Unvorhersagbarkeit und Varianz interner Parameter, wie z. B. der Anzahl und Länge von Menüeinträgen auf der einen Seite und der durch die Anforderung der technischen Niedrigschwelligkeit bedingte Varianz der Darstellung auf der anderen Seite, musste zunächst ein flexibles Grundlayout geschaffen werden, dessen Funktion und Wiedererkennbarkeit möglichst auch bei extremen Werten der obigen Parameter gewährleistet ist. Als grundsätzliche Konsequenz ergab sich daraus ein einfaches Layout, das die Unvorhersehbarkeit seiner genauen Darstellung im jeweiligen Browser des Benutzers so weit wie möglich mit einkalkuliert, d. h. nicht von einer vollständigen pixelgenauen Kontrollierbarkeit seiner Darstellung ausgeht.

Eine spezifische Konsequenz aus den besonderen Anforderungen war z. B. der Verzicht auf die horizontale Anordnung einiger Navigationselemente. Insbesondere die *Brotkrumenliste* (Abbildung 5.4 auf Seite 132 ❹) wurde nicht wie sonst üblich horizontal, sondern vertikal dargestellt, da sich sonst eine

unübersichtliche und zu viel »Inhaltsraum« einnehmende mehrzeilige Anzeige nicht hätte ausschließen lassen. Bei den vertikal angeordneten Navigationseinträgen musste dagegen z. B. ihre – aufgrund ihrer nicht vorhersagbaren Länge – nicht auszuschließende mehrzeilige Darstellung bei der Konzeption des Layouts berücksichtigt werden.

Um den einzelnen Einrichtungen und Fakultäten unter dem Corporate Design (CD) der Universität eigene »*Corporate Sub-Identitities*« ermöglichen, ohne dabei den Wiedererkennungseffekt der Site zu gefährden, wurde das Layout außerdem dahingehend modular konzipiert, dass sich die Modifikation bestimmter Elemente zur Hervorhebung eigener Identitäten anbietet. Für diesen Zweck waren insbesondere das Logo links oben (Abbildung 5.4 auf Seite 132 ❹) – für das die Spezifikation des CD der Universität bereits Abwandlungen reglementiert– sowie Titelschriftzug und Ausklappmenüleiste (Abbildung 5.4 auf Seite 132 ❶) vorgesehen. Die Möglichkeiten der Modifikation waren zwar nicht auf diese Elemente beschränkt[8], dem Vorschlag wurde jedoch durch die Dokumentation zur Realisierung von *Sub-Identities* und entsprechende Beispielvorlagen zusätzlich Nachdruck verliehen. Außerdem wurden, um auch für ein einheitliches Erscheinungsbild des Inhaltsbereichs Sorge zu tragen, Gestaltungsregeln auf den Webseiten des Informationsmanagements veröffentlicht.

### 5.4.2 Struktur und Navigation

Ein großer Teil der Planungsarbeiten für den neuen Webauftritt der Universität Bielefeld bestand darin, eine Struktur für alte und neu hinzukommenden Inhalte der Site zu finden. Im Gegensatz zu der ungesteuert und unkoordiniert gewachsenen Struktur des alten Auftritts, sollte die neue ein logisch nachvollziehbares Gerüst bilden, in dem sich sowohl Autoren als auch die verschiedenen Besuchergruppen leicht zurechtfinden.

[8] siehe Abschnitt 5.5.2 zur technischen Realisierung von *Sub-Identities*

#### 5.4.2.1 Hauptnavigationsstruktur und Startseite

Grundlage der neuen Navigationsstruktur ist die bereits in der Vorbereitungsphase beschlossene Aufteilung in eine an der Organisationsstruktur orientierte und eine benutzerspezifische Gliederung (siehe Abbildung 5.2 , »Universität« bzw. »Benutzer«). Durch diese Aufteilung soll sowohl den mit der Universität vertrauten Studenten und Bediensteten als auch den von außen kommenden unterschiedlichen Interessentengruppen ein schnelles Auffinden von Informationen ermöglicht werden. Erweitert wurde diese ursprüngliche Gliederung der ersten drei Hierarchieebenen um einen Zweig für unterschiedliche ausländische Interessentengruppen (Abbildung 5.2 , »International«).

Diese Hauptnavigationsstruktur findet sich ebenso wie die farbliche Kodierung der drei Hauptzweige sowohl in Navigationselementen der zentralen Seiten (siehe Abschnitt 5.4.2.3) als auch auf der Startseite der Universität (Abbildung 5.3 auf Seite 130) wieder. Auf der Startseite befinden sich dabei zusätzlich Direkteinsprünge zu Seiten, die zusätzlich zur hierarchischen Navigation, das direkte Finden von Informationen über eine interne Suchmaschine, eine A-Z-Liste und telefonischen oder postalischen Kontakt ermöglichen.

#### 5.4.2.2 Isomorphie von Dateisystem und Hauptnavigationsstruktur

Als technische Umsetzung der Navigationsstruktur der ersten drei und der folgenden Ebenen wurde ihre Eins-zu-Eins-Umsetzung in das Dateisystem des CMS gewählt. Das bedeutet, dass die neu geplante hierarchische Baumstruktur des Auftritts direkt in die hierarchische Baumstruktur des Dateisystems (siehe Abbildung 5.2 ) abgebildet ist. Um dabei verzweigende Knoten mit ausreichend Informationen versehen zu können, sind diese nicht direkt durch Ordner repräsentiert, sondern durch deren jeweilige Index-Dateien. Index-Dateien nehmen also die Rolle einer Übersicht oder einer Zusammenfassung über die Inhalte ihres Ordners bzw. des von ihnen ausgehenden Baums ein.

Der Vorteil der Isomorphie von Ordnerstruktur und Benutzernavigation liegt vor allem in der Vermeidung einer redundanten Haltung und Wartung

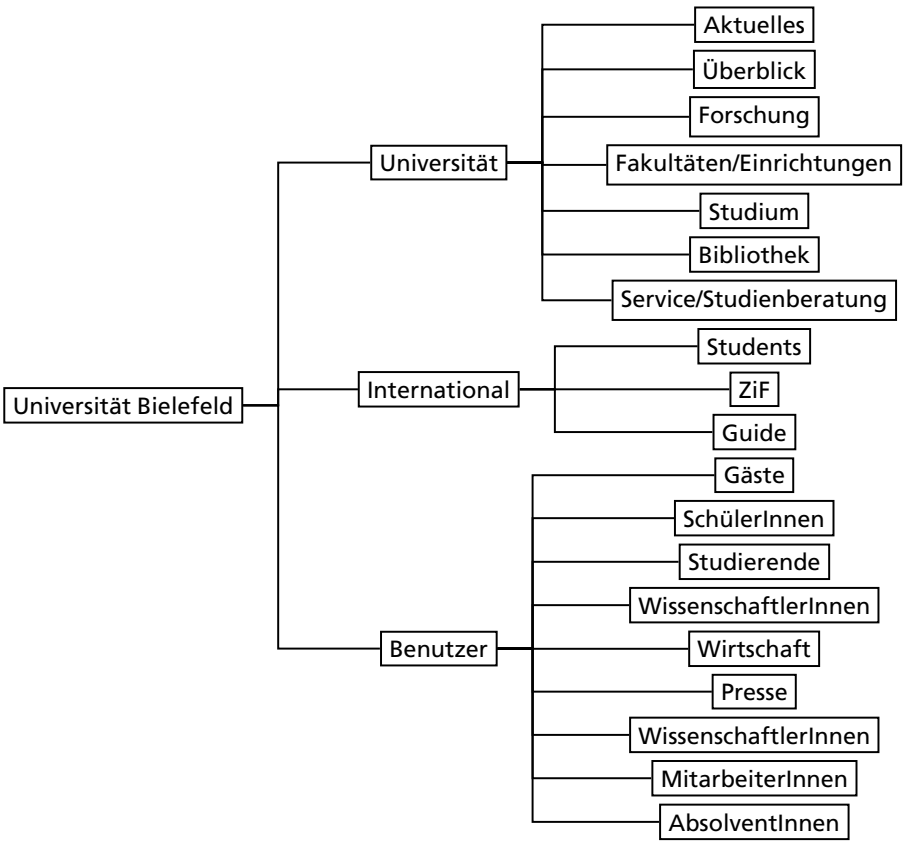


**Abbildung 5.2:** *Die ersten drei Hierarchieebenen des neuen Webauftritts der Universität Bielefeld.*


**Abbildung 5.3:** Die neue Startseite der Universität Bielefeld

von Navigationselementen. Autoren können wie gewohnt ihre Dateien in Ordnern organisieren. Die dieser Organisation entsprechenden Navigationselemente können, wenn erwünscht, automatisch auf Grundlage des Dateisystems und Metainformationen von Stylesheets erzeugt werden. Autoren sind so von der Gestaltung und Programmierung von Navigationselementen und deren Synchronisation mit der tatsächlichen Dateisystemstruktur entlastet. Fehlende und auf nicht mehr existente Seiten verweisende Links sind ausgeschlossen, da genau die vorhandenen, sichtbaren Dateien anzeigt werden. Jede Änderung im Dateisystem ist sofort in den Navigationselementen sichtbar. Der Aufwand für die Wartung der Navigation und die Kontrolle auf fehlende und fehlerhafte interne Links entfällt somit auch. Letzteres bedeutet insbesondere dann eine große Arbeitserleichterung, wenn eine Site von mehreren Autoren bearbeitet wird und das Einstellen oder Löschen von Seiten ohne eine Automatisierung jedes Mal allen betroffenen Autoren mitgeteilt werden müsste, damit diese ihre Links aktualisieren.

Durch die direkte Abbildung ihrer Ordnerstruktur in die Benutzernavigation sind Autoren außerdem dazu angehalten, zunächst ein tragfähiges Hierarchiegerüst für ihre Site zu planen und ihre Inhalte von vornherein

systematisch in Ordnern und Unterordnern abzulegen. Der dadurch verursachte anfängliche Mehraufwand erspart aber, neben den bereits erwähnten Wartungsarbeiten, das »Verlorengehen« von Dateien und spätere Umstrukturierungen der Site und daraus resultierende fehlerhafte Links und veraltete Bookmarks auf der Benutzerseite (vgl. Abschnitt 4.9).

Die Gleichsetzung von Ordner- und Navigationsstruktur impliziert allerdings nicht die Reduktion der Traversierbarkeit der Site auf eine einfache Baumstruktur. Andere Vernetzungen können über die Inhalte zum einen natürlich weiterhin durch Links innerhalb der Dokumente gelegt werden. Zum anderen besteht die Möglichkeit, durch *Redirects* oder eigene Menübeschreibungen (siehe Abschnitt 5.4.2.4) auf andere Punkte in der Verzeichnishierarchie zu verweisen. Solche Verweise schlagen sich – bei automatischer Generierung – auch in den Navigationselementen nieder. Dabei ist allerdings zu beachten, dass unter Umständen bereits wenige solcher Querverweise ausreichen, um die ansonsten einfache Baumstruktur der Hauptnavigation in Bezug auf ihre *Verlässlichkeit* folgenschwer zu untergraben. Denn wenn die Verlässlichkeit der Baumstruktur nicht gegeben ist, kann ein Besucher kaum ein kognitives Modell der Site-Struktur aufbauen – was wiederum ein »Zurechtfinden« stark beeinträchtigt. Eine Möglichkeit, diesem Problem zu begegnen, wäre vermutlich, solche Querverweis-Links grafisch zu kennzeichnen. Zur Zeit ist dies allerdings noch nicht implementiert.

*Direkteinsprünge* auf beliebige Punkte der Sitehierarchie können außerdem durch Erweiterungen der im Folgenden beschriebenen Ausklappmenüs realisiert werden.

#### 5.4.2.3 Die Hauptmenüleiste

Die *Hauptmenüleiste* rechts oben (Abbildung 5.4 ❶) soll, unabhängig von der Position der aktuellen Seite in der Hierarchie, den schnellen und direkten Zugriff auf Top-Level- und andere wichtige Seiten erlauben. Dieses Menü ist also *kontextunabhängig*, d. h. über viele oder alle Seiten und Hierarchieebenen identisch. Der permanent sichtbare Teil der Leiste enthält standardmäßig die drei von der Startseite her bekannten Oberbegriffe der verschiedenen Zugänge: *Universität*, *International*, *Benutzer* und zusätzlich ein Suche-Symbol

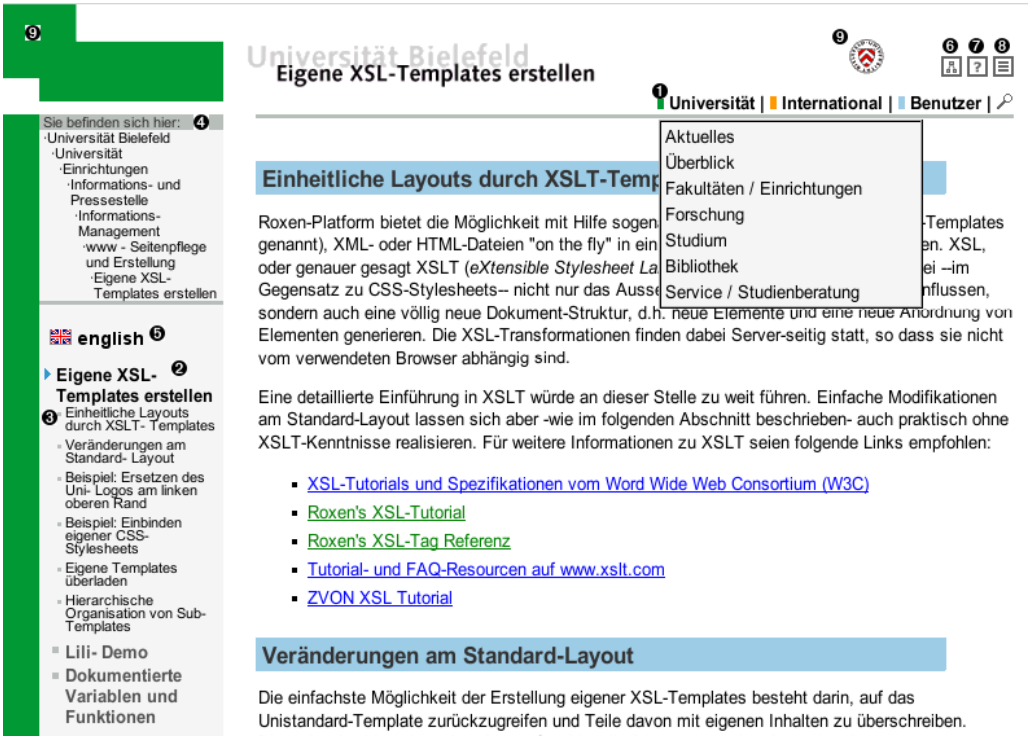


**Abbildung 5.4:** Die Standardnavigationselemente der Uni-Seiten.
❶ Hauptmenüleiste (Top-Level- und andere wichtige Seiten), ❷ Navigationsmenü (lokale Navigation im aktuellen Ordner), ❸ Inhaltsverzeichnis (Einsprünge innerhalb der aktuellen Seite), ❹ Brotkrumenliste (Pfad innerhalb der Hierarchie vom aktuellen Ordner zur Startseite), ❺ Auswahl alternativer Sprachvarianten des aktuellen Dokuments (falls vorhanden), ❻ Link auf Sitemap (Hierarchiebaum ab der aktuellen Ebene), ❼ Link auf Hilfe-Seite, ❽ Link auf Druckversion des aktuellen Dokuments, ❾ Logoelemente mit Link auf Startseite der Site oder Sub-Site.

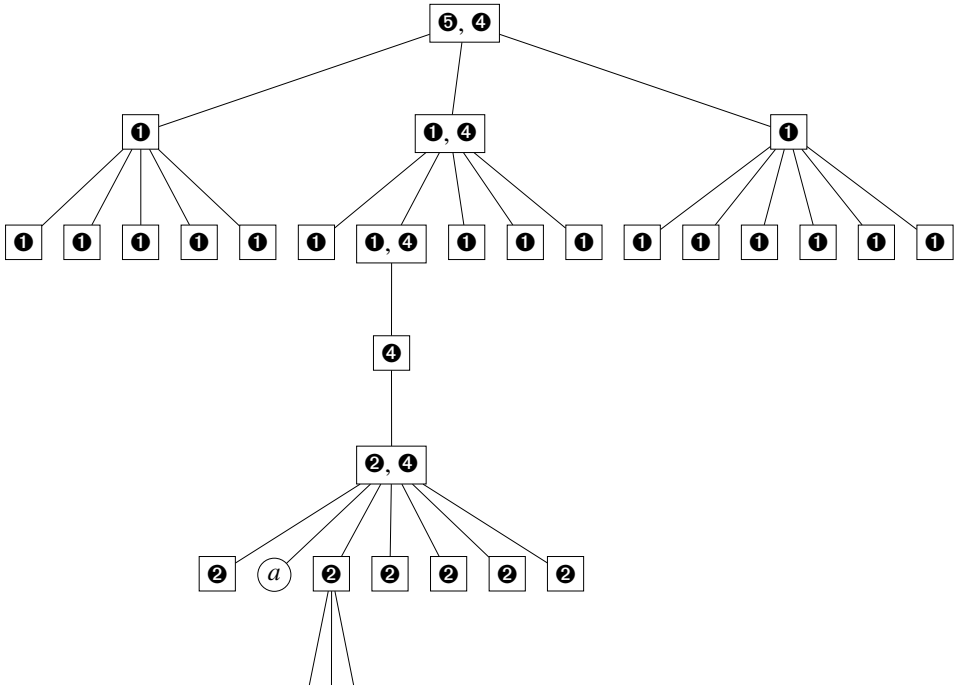


**Abbildung 5.5:** *Über die Standardnavigation direkt von einer Seite a erreichbare Seiten..* Dargestellt ist lediglich der direkt erreichbare Teil der Hierarchie. Die Nummern repräsentieren dabei die Navigationselemente mit Hilfe derer auf die zugehörige Seite gesprungen werden kann: ❶ Hauptmenüleiste, ❷ lokales Navigationsmenü, ❹ Brotkrumenliste, ❾ Logoelemente. (siehe Abbildung 5.4 ).

als Oberbegriff für besonders häufig frequentierte (»gesuchte«) Seiten.

Die Einträge der zum jeweiligen Oberbegriff gehörenden Ausklappmenüs entsprechen ebenfalls denen der Startseite.

Die automatische Generierung der Ausklappmenüleiste erfolgt zur Vermeidung von Redundanzen anhand der selben XML-Beschreibungsdatei[9] wie die Generierung der Startseitenmenüs. Die Beschreibungsdatei legt z. B. auch fest, ob Einträge eventuell nur auf der Startseite oder nur im Menü erscheinen sollen und welche Icons jeweils verwendet werden.

In eigenen Abwandlungen des Uni-Stylesheets kann durch die Festlegung einer oder mehrerer weiterer Beschreibungsdateien dieser Art die Hauptmenüleiste z. B. um die Hauptnavigation einer Einrichtung oder Fakultät erweitert werden (siehe Beispiel Sportwissenschaften[10] und `pdmenu-file`-Dokumentation[11]).

Auf eine vollautomatische Generierung des Hauptmenüs anhand des Dateisystems wurde deshalb verzichtet, weil seine Wichtigkeit eine redaktionelle Bearbeitung (z. B. die Sortierung der Einträge und das Einfügen von nicht der Ordnerhierarchie entsprechenden *Direkt-Links*) notwendig macht.

#### 5.4.2.4 Das kontextabhängige Navigationsmenü

Das Navigationsmenü links (Abbildung 5.4 auf Seite 132 ❷) soll den Zugriff auf Seiten erlauben, die in der Hierarchie »nah« an der aktuellen Seiten liegen, also gemäß der gewählten Hierarchisierung mit ihr verwandt sind. In Bezug auf das Dateisystem ausgedrückt, werden im Navigationsmenü die Dateien angezeigt, die im Ordner der aktuellen Datei liegen. Der Titel der Index-Datei des Ordners fungiert dabei als Überschrift und wird oben fett und nach links eingerückt angezeigt. Der Titel der aktuell angezeigten Seite ist durch einen Pfeil markiert. Auf die Darstellung der Hierarchieebenen direkt oberhalb oder unterhalb der aktuellen wurde nach vielen Tests verzichtet. Es hat sich herausgestellt, dass die Menüs dadurch oft zu lang und damit unübersichtlich werden. Außerdem wird der Benutzer bereits durch

[9] `view-source:http://www.uni-bielefeld.de/etc/unibiMenu.xml`
[10] `http://www.uni-bielefeld.de/sport/arbeitsbereiche.html`
[11] `http://www.uni-bielefeld.de/Universitaet/Einrichtungen/Pressestelle/Informationsmanagement/Help/Templates/einfach.html?variable=pdmenu-file`

die »Brotkrumenliste« (siehe ❹) darauf hingewiesen, *wo* er sich innerhalb der Hierarchie befindet. Auch auf Ausklappmenüs, um die Inhalte von Ordnern bei der Berührung mit der Maus anzuzeigen, wurde an dieser Stelle zunächst verzichtet, da zu viele dynamische Elemente ebenfalls eher Verwirrung stiften, als dass sie die Navigation erleichtern.

Die Generierung des Navigationsmenüs erfolgt standardmäßig automatisch. Das Uni-Stylesheet listet die Titel aller im aktuellen Verzeichnis befindlichen Ordner (bzw. die Titel ihrer Index-Dateien) und alle sichtbaren Dateien der Typen *html*, *xhtml* und *xml* in alphabetischer Reihenfolge. Falls keine vollautomatische Menügenerierung gewünscht ist, kann im Verzeichnis eine spezielle Menüdatei (Extension `.menu`) angelegt werden, die die Namen der gewünschten Dateien und Order, mit den dazugehörigen Titeln, in der gewünschten Reihenfolge beinhaltet.

**Inhaltsverzeichnis**

Zur dokumentinternen Navigation wird unterhalb des angewählten Eintrags im Navigationsmenü außerdem eine Art Inhaltsverzeichnis der Seite dargestellt. Standardmäßig sind dort alle `H1`-Überschriften – sofern mehr als eine vorhanden ist – mit entsprechenden Links hinterlegt, aufgelistet. Welche Elemente ins Inhaltsverzeichnis aufgenommen werden, ist in eigenen Stylesheets konfigurierbar (`toc-entry`-Dokumentation[12]).

#### 5.4.2.5 Brotkrumenliste

Die grundlegende Voraussetzung für die Navigierbarkeit einer Web-Site ist, dass der Benutzer weiß, wo er sich gerade befindet. Diese wichtige Frage nach dem »Wo bin ich?« soll durch das mit »*Sie befinden sich hier*« betitelte Navigationselement (siehe Abbildung 5.4 auf Seite 132 ❹) beantwortet werden. Dort ist der Pfad von der Startseite zum aktuellen Knoten im Baum anhand der auf dem Weg liegenden Knoten (bzw. Ordnern), wie mit Brotkrumen markiert. Dem Benutzer wird mit dieser sogenannten *Brotkrumenliste* [Nie00], unterstützt durch die Einrückungen, das Verständnis für den hierarchischen Aufbau der Site und das Verständnis der Bedeutung der aktuellen Seite durch

[12] `http://www.uni-bielefeld.de/Universitaet/Einrichtungen/Pressestelle/Informationsmanagement/Help/Templates/einfach.html?variable=toc-entries`

die Einbettung in ihren hierarchischen Kontext erleichtert. Außerdem hat er die Möglichkeit, falls die aktuelle Seite nicht alle von ihm gesuchten Informationen beinhaltet, direkt auf höhere Ebenen mit entsprechend allgemeineren Informationen zu wechseln und von dort aus weiterzusuchen.

#### 5.4.2.6 Sitemap

Als weitere Möglichkeit zur hierarchieorientierten Navigation, wird von jeder Seite ein Link (Abbildung 5.4 auf Seite 132 ❻) auf eine – einen Überblick über den aktuellen Teil der Site vermittelnde – Sitemap (siehe Abbildung 5.6 ) angeboten. Diese Sitemaps sind partiell und kontextsensitiv, d. h. es wird nur der Teil des Hierarchiebaums angezeigt, der sich im Dateisystem unterhalb des Ordners befindet, aus dem heraus die Sitemap aufgerufen wurde. Grundsätzlich eine komplette Karte anzuzeigen ist aufgrund der Menge der Seiten der Universität nicht sinnvoll. Dementsprechend werden Sitemaps standardmäßig auch nicht direkt von Hand erstellt, sondern von einem speziellen XSLT-Stylesheet anhand von Metadaten und Dateisysteminformationen automatisch generiert.

#### 5.4.2.7 Sprachvarianten-Auswahl

Da Roxen die Verwaltung von Sprachvarianten beliebiger Ressourcen, ähnlich wie in Abschnitt 4.8 beschrieben, unterstützt, konnte der Umgang mit in Übersetzungen vorliegenden Dokumenten in allen Punkten wie geplant realisiert werden.

Die Auswahl der Sprachvariante einer Seite erfolgt standardmäßig anhand der vom Besucher im Browser eingestellten Präferenzliste. Diese Einstellung beeinflusst nicht nur die Wahl der Dokumentvariante und Grafikvarianten im Seitenkörper, sondern auch die Beschriftungen und Hilfstexte zu Navigationselementen sowie die Meldungen im Fuß der Seite. Auch dann, wenn das Dokument selbst nicht in Übersetzungen vorliegt. Falls ein Dokument in einer laut seiner Metadaten nicht aktuellen Übersetzung angezeigt wird, wird darauf durch einen entsprechende Hinweis im Seitenkopf hingewiesen. Zusätzlich zur automatischen Wahl der Sprachvariante besteht – unter anderem deshalb – die Möglichkeit, andere Sprachvarianten – sofern vorhan-


**Abbildung 5.6:** Ausschnitt aus der Sitemap: Einrichtungen

den – entweder durch einen automatisch erzeugten Link (Abbildung 5.4 auf Seite 132 ❺) oder die direkte Eingabe eines Pfadpräfix im URI – wie z. B. `/(en)/index.html` für die englische Variante der Index-Seite – explizit auszuwählen.

Die Kodierung einer explizit gewählten Sprache als Pfadpräfix hat außerdem den Vorteil, dass erstens die Auswahl bei relativer Verlinkung für die weitere Site-interne Navigation bestehen bleibt und zweitens, dass bestimmte Sprachvarianten mit Lesezeichen versehen und verlinkt werden können.

#### 5.4.2.8 Druckfassung

Da Roxen über keinen FO-Formatter[13] verfügt, wurde auf eine direkte Generierung von PDF-Ausgaben zunächst verzichtet. Stattdessen ist das standardmäßig auf allen Seiten vorhandene Druckfassungssymbol (Abbildung 5.4 auf Seite 132 ❽) mit einer durch den Browser auszudruckenden besonderen HTML-Ansicht der aktuellen Seite verlinkt. Diese Ansicht wird durch eine Spe-

[13] Siehe Abschnitt 2.5 zu XSL *Formatting Objects*

zialisierung[14] des zur Generierung der normalen Webansicht verwendeten XSLT-Stylesheets erzeugt. Sie unterscheidet sich normalerweise von der Webansicht vor allem durch das Fehlen von Navigationselementen.

#### 5.4.2.9 Suche

Die interne Suchmaschine der Universität ist über das Suchmenü (Abbildung 5.4 auf Seite 132 ❶ rechts) zu erreichen. Da sich im Rahmen der Migration herausstellte, dass die CMS-eigene Suchmaschine aus Performanzgründen nicht einsetzbar war, wurde zunächst die bereits zuvor verwendete Volltext-Suchmaschine der Firma Verity weiterverwendet. Weil diese aber sowohl was die Suchmöglichkeiten als auch die Bewertung der Treffer angeht nicht überzeugen konnte, wurde der Suche im Standardlayout der Seiten zunächst keine prominentere Stelle eingeräumt.

Zur Zeit wird als Übergangslösung die vom *Zentrum für Lehrerbildung* betriebene freie Suchmaschine mnoGoSearch[15] eingesetzt. In nächster Zeit soll jedoch auf eine neue Verity-Version umgestellt werden. Diese wird dann so konfiguriert sein, dass sie einem inzwischen erstellten Anforderungskatalog genügt und z. B. eine gezielte Suche nach Autorennamen oder anderen Metadaten ermöglicht.

## 5.5 Implementation

Da Roxen sowohl XSL-Transformationen als auch eine eingebettete Scriptsprache (RXML) unterstützt, konnte im Wesentlichen eine wie in Kapitel 2 beschriebene grundsätzlich zweistufige Architektur zur Aufbereitung von Inhalten realisiert werden (siehe auch Abbildung 5.7 auf Seite 141):[16]

1. Stufe: Ein in HTML oder XML-Sprachen vorliegendes Dokument mit eventuellen RXML-Einbettungen wird anhand eines entsprechend der

[14] Siehe Abschnitt 2.4 zur Realisierung von Spezialisierungen bzw. Klassenhierarchien in XSLT.

[15] `http://search.mnogo.ru/`

[16] Der prinzipiell zweistufige Aufbereitungsprozess wird an einigen Stellen durch Verkettung von XSL-Transformationen und vorgeschaltete Übersetzung bestimmter RXML-Code-Fragmente erweitert.

Ausgangssprache und des gewünschten Layouts voreingestellten XSLT-Stylesheets in HTML mit eventuellen RXML-Einbettungen transformiert. Bei dieser Transformation hat das Stylesheet bzw. der XSLT-Prozessor außer dem Ausgangsdokument über eine XPath-Erweiterungsfunktion Zugriff auf URI-Parameter, Metadaten, sowie Dateisysteminformationen.

2. Stufe: Die Anweisungen des eingebetteten RXML-Codes werden in entsprechendes HTML und evtl. zusätzliches Javascript und Grafiken übersetzt. Von RXML aus besteht dabei gegenüber den XSLT-Stylesheets zusätzlich Zugriff auf Datenbanken und HTTP-Anfrageparameter. Außerdem können Grafiken und HTTP-Antwort-Header-Angaben generiert werden.

Um die Abhängigkeiten vom CMS möglichst gering zu halten, wurden Funktionalitäten dabei so weit wie möglich in transportablen XSLT-Stylesheets implementiert. Insbesondere zur automatischen Generierung von Navigationselementen musste allerdings auf die Roxen-eigene XPath-Erweiterungs-Funktion `rxml:metadata(`*Path*`)` zurückgegriffen werden. Abhängig vom Pfadargument berechnet diese Funktion eine XML-Struktur, die entweder Metadaten zu einer Datei (z. B. Pfad, URL, Autor, Modifikationsdatum, Schlüsselwörter, Beschreibung, ...) oder die Inhaltsangabe eines Verzeichnisses enthält. Im Einzelnen wurde `rxml:metadata` zur Generierung der Brotkrumenliste, des Navigationsmenüs, sowie zur Erzeugung von Informationen zu Autor und Modifikationsdatums und zur Generierung von *Dublin-Core*-Metadaten (siehe Abschnitt 5.6.2 zur Implementation) verwendet.

Die in der zweiten Transformationsstufe übersetzte Scriptsprache RXML dient im Standard-Stylesheet zum einen zur Erzeugung von HTTP-Headern[17] und zum anderen zur Transformation und zur Erzeugung von Grafiken. Mit Hilfe der RXML-Anweisung `gtext` werden z. B. Titelgrafiken der Uni-Seiten[18] anhand von Variablen, Metadaten sowie ebenfalls einstellbaren Angaben zu Font, Größe und Farbe *on-the-fly* vom RXML-Prozessor erzeugt. Die so ermöglichte Entkoppelung von Inhalt und

[17] im Einzelnen: Last-Modified, Expires und Vary

[18] siehe z. B. Abbildung 5.10 auf Seite 147, »Universität Bielefeld« bzw. »Eigene XSL-Templates erstellen«

| **Stylesheet/ Script** | **Funktion/ Inhalt** |
|---|---|
| `uni.xsl` | Standard-Stylesheet der Universität Bielefeld. Importiert mittelbar oder unmittelbar alle weiteren namentlich aufgeführten XSLT-Stylesheets. |
| `param.xsl` | Enthält die Instantiierungen aller Parameter und Variablen die im Standard-Stylesheet verwendet werden. |
| `navigation.xsl` | Verantwortlich für die Erzeugung aller Navigationselemente (siehe Abschnitt 5.4.2). |
| `pdmenu.xsl` | Generiert Ausklappmenüs (HTML+Javascript). Wird vom Standard-Stylesheet zur Erzeugung der Hauptmenü-Leiste (Abbildung 5.4 auf Seite 132 ❶) verwendet. |
| `html-treatment.xsl` | Durch HTML-Elemente getriggerte Regeln: zur Dokumentkorrektur (z. B. zur Einbettung »elternloser« Textknoten in `p`-Elemente) und *Rendering* bestimmter HTML-Elemente (z. B. farbliche Markierung interner Links). |
| `unibi-elements.xsl` | Regeln zur Verarbeitung spezieller Elemente im `unibi`-Namensraum (z. B. `unibi:toc-entry` zur Erzeugung von Inhaltsverzeichniseinträgen). |
| `metadata.xsl` | Generiert Metadaten für das `head`-Element und Link auf RDF/XML-Metadaten. |
| `sitemap.xsl` | Standard-Stylesheet zur Generierung von *Sitemaps* (siehe Abschnitt 5.4.2.6). |
| `tunnel.xsl` | Verantwortlich für die gepufferte Einbindung von Ressourcen externer HTTP-Server (siehe Abschnitt 5.6.3). |
| `rdf.xsl` | Transformiert die Metadaten einer Ressource in RDF/XML-Aussagen (siehe Abschnitt 5.6.2). |
| `print.xsl` | Standard-Stylesheet zur Generierung von Druckversionen (siehe Abschnitt 5.4.2.8). |
| `uni.xsl`-*Spezialisierungen* | Spezielle Stylesheets von Fakultäten und Einrichtungen zur Realisierung von *Sub-Identities* und speziellen Funktionalitäten (siehe Abschnitt 5.5.2). |
| `xbel.xsl`, `docbook.xsl`, `tei.xsl`, ... | Transformation der entsprechenden Standard-XML-Dokumentklassen (XBel, Docbook, TEI, ...) nach HTML. |
| `tunnel-update.pl` | Perl-Script zum Auffrischen des Datenbankpuffers der für die Einbindung von Ressourcen externer HTTP-Server verwendet wird (siehe Abschnitt 5.6.3). |
| `Popup.js` | Javascript-Funktionen zum Handling von Ausklappmenüs. |

**Tabelle 5.2:** *An Generierung und Darstellung der Universitätsseiten beteiligte Stylesheets und Scripts..*
Die Quelltexte sind unter `http://www.uni-bielefeld.de/templates/` bzw. `http://www.uni-bielefeld.de/javascript/` frei zugänglich.

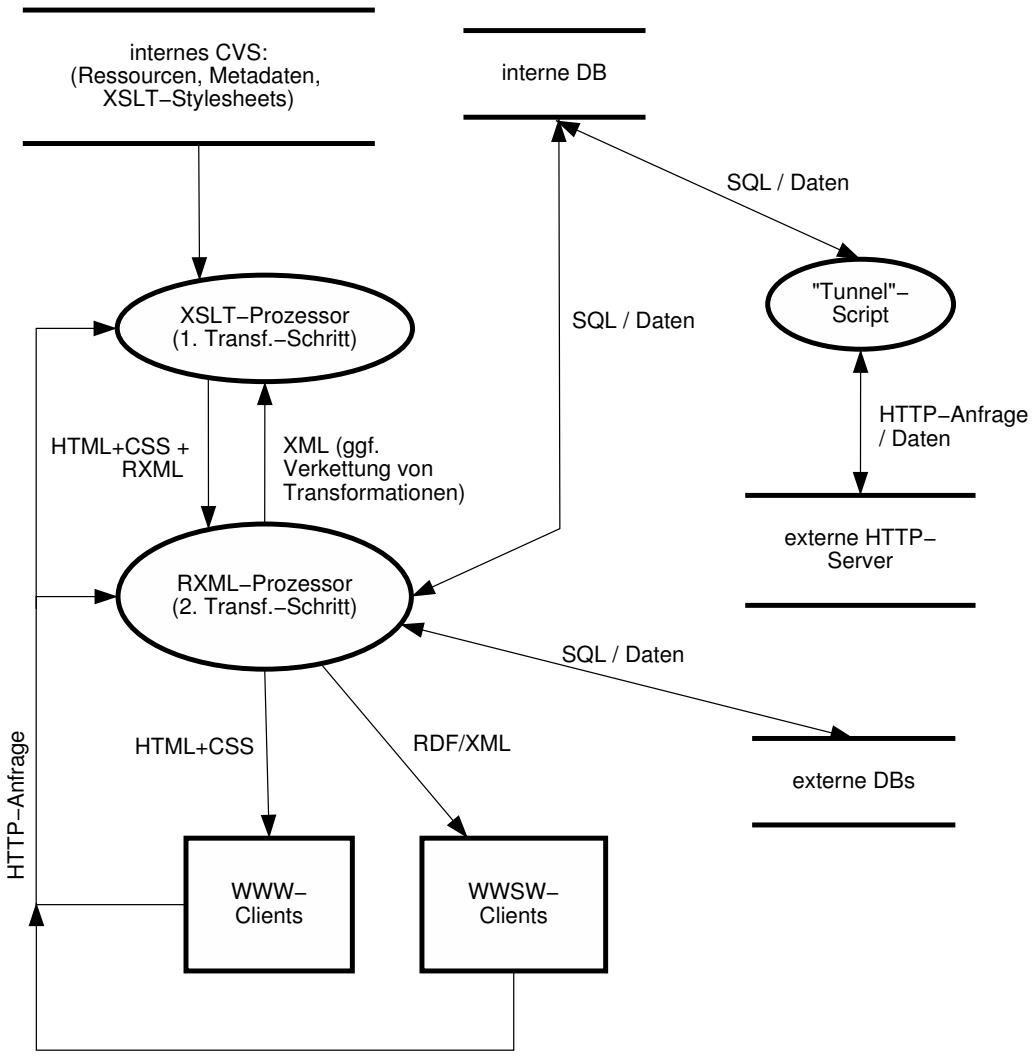


**Abbildung 5.7:** Implementierte Datenfluss-Architektur zur Aufbereitung von Inhalten für Web und Semantic Web

Gestaltung erspart gerade im Zusammenhang mit solchen Textgrafiken sonst notwendigen erheblichen Erstellungs- und Wartungsaufwand und sorgt für ein einheitliches und flexibel modifizierbares Erscheinungsbild.

### 5.5.1 Caching

Insbesondere auch zur Pufferung aufwendiger dynamisch erzeugter Grafiken werden Roxens Fähigkeiten zum Cachen von Transformationsresultaten, ähnlich wie in Abschnitt 4.4.2 beschrieben, verwendet. Das bereits in der ersten Transformationsphase interpretierte, Roxen-spezifische XSLT-Erweiterungselement `cache` erlaubt die Speicherung seiner transformierten und interpretierten Inhalte und deren Wiederverwendung über beliebige Dokumente und Stylesheets hinweg. Der zur Speicherung und Aufruf eines Caches benötigte *Schlüssel* wird dabei defaultmäßig über den untransformierten Inhalt des `cache`-Elements gebildet. Es ist jedoch auch möglich, den Schlüssel explizit durch das Attribut key zu setzen. Das Element, das den Hintergrundschriftzug – in der Regel »Universität Bielefeld« – des Seitentitels generiert, sieht etwas vereinfacht z. B. folgendermaßen aus:

```
<cache shared="shared" key="tbgt-{$title-bgtext}">
  <gtext-url fontsize="23" bold="bold" fgcolor="#cccccc"
             bgcolor="white" font="syntax">
    <xsl:value-of select="$title-bgtext"/>
  </gtext-url>
</cache>
```

Auf diese Weise werden außerdem standardmäßig die über jeweils viele Seiten identischen, in ihrer Generierung aufwendigen Ausklappmenüs (Abbildung 5.4 auf Seite 132 ❶), sowie das Logoelement ( ❶ links oben), das in seiner Breite dem Menübalken angepasst wird, gecacht.

### 5.5.2 Realisierung von Corporate Sub-Identities

Eine wesentliche Anforderung an die Content-Aufbereitung war, dass Einrichtungen und Fakultäten einerseits angehalten sein sollten, sich am *Corporate Design* der Universität zu orientieren, andererseits aber auch die Möglichkeit haben sollten, ihre eigene Identität grafisch zum Ausdruck zu bringen. Analog dazu sollten auch Projekte und einzelne Studiengänge wiederum in der Lage sein, das Layout ihrer übergeordneten Fakultäten ihren eigenen Bedürfnissen entsprechend, möglichst einfach

modifizieren zu können.

Das grundlegende Konzept zur technischen Realisierung dieser Anforderungen ist eine Vererbungshierarchie von XSLT-Stylesheets mit dem Standard-Stylesheet der Universität als Basis. Das bedeutet, die Standardvorgehensweise zur Realisierung einer sogenannten *Corporate Sub-Identity*, z. B. für eine Fakultät, ist die Einrichtung eines XSLT-Stylesheets, welches das Standard-Stylesheet importiert und bestimmte Variableninitialisierungen, Funktionen oder Regeln durch eigene überschreibt. Der Vorteil einer solchen *Spezialisierungsstrategie* gegenüber einfachem Kopieren und Verändern ist, dass keine unerwünschten Redundanzen eingeführt werden und damit der Aufwand für Wartung und Weiterentwicklung insgesamt auf das notwendige Maß reduziert bleibt.

Der Vorteil gegenüber einer durch das Einbinden von einzelnen Modulen gekennzeichneten *Bibliotheks-* oder *Warenkorbstrategie* besteht darin, dass Spezialisierungen innerhalb einer Klassenhierarchie automatisch auch von neuen Modulen ihrer übergeordneten Klassen profitieren, ohne dass sie selbst verändert werden müssen. Darüber hinaus ist eine ausreichend fein granulierte dateibasierte Modularisierung von Stylesheets und die Berücksichtigung der Abhängigkeiten zwischen diesen schwierig und sehr umständlich. Aus diesen Gründen wird eine dateibasierte Modularisierung allein dazu verwendet, um ein selektives Ererben bzw. Nicht-Ererben von respektive:

- grundlegenden Funktionen (z. B. zur Erzeugung von Ausklappmenüs (siehe Abschnitt 5.4.2.3 bzw. Abbildung 5.4 auf Seite 132 ❶)
- für das Rahmen-Layout und Navigationselemente verantwortlichen Funktionen und Templates
- für die Transformation von Dokumentinhalten verantwortliche Templates

zu ermöglichen (siehe Tabelle 5.2 auf Seite 140). Insbesondere ein selektives *Nicht-Ererben* letzterer, von Dokumenten getriggerter Templates bzw. Regeln kann notwendig sein, z. B. um bestimmte standardmäßig vorgenommene HTML-Korrekturen zu umgehen oder bestimmte, nicht vom Standard-Stylesheet unterstütze XML-Sprachen zu transformieren, ohne dabei jedoch auf Standardfunktionalitäten und Standardlayout zu verzichten.

### 5.5.3 Modularisierung

Voraussetzung einer flexiblen Modifizierbarkeit von Layout und Elementen durch spezialisierte Stylesheets war eine entsprechend feine Modularisierung der Parameter und Funktionen des Standard-Stylesheets. Die Strategie dabei war, zunächst möglichst alle Zeichenketten, Farben und Maßangaben der Layoutvorgabe als solche zu parametrisieren oder – bei abhängigen Variablen – diese durch Funktionen zu beschreiben. Außerdem wurde das Stylesheet zum einen entsprechend der grafischen und zum anderen entsprechend der generischen Struktur der Layoutvorgabe in sinnvoll erscheinende Einheiten (Funktionen, Templateregeln und Variablen) aufgeteilt. Diese doppelte Aufteilung war notwendig, um Spezialisierungen zu erlauben, Elemente einerseits anhand ihrer Art oder Funktion (z. B. »Hauptmenüleiste«), unabhängig von ihrer aktuellen oder zukünftigen Position im Layout, verändern zu können und andererseits Elemente explizit anhand ihrer Position im Layout (z. B. »*header-right-bottom*«) ohne Rücksicht auf deren Funktion austauschen zu können. Die daraus resultierende hierarchische Struktur des Stylesheets war auf oberen Ebenen durch sowohl generische als auch auf das Seitenlayout bezogene Verzweigungen (z. B. »header« und »footer«) geprägt, während untere Ebenen hauptsächlich generisch aufgeteilt waren (z. B. »menu-entry« oder »contact-link«).

### 5.5.4 Dokumentation

Um Webmastern oder anderen Vertretern von Einrichtungen die Anpassung von Stylesheets an ihre eigenen Bedürfnisse zu erleichtern, wurden alle in Frage kommenden Variablen des Standard-Stylesheets mit speziellen Attributen typisiert und dokumentiert. Anhand dieser Attribute im RXML-Namensraum können die unabhängigen Variablen über eine Roxen-eigene Webschnittstelle – auch für Stylesheets die diese Variableninstantiierungen lediglich importieren – leicht modifiziert werden (siehe Abbildung 5.8 ).

Um außerdem die Veränderung berechneter Variablen und das Überschreiben von Funktionen und Regeln zu erleichtern, wurde ein spezielles XSLT-Stylesheet entwickelt, welches anhand der aktuellen Version des Standard-Stylesheets automatisch eine dokumentierte Liste von zum Überschreiben freigegebenen Elementen generiert (Abbildung 5.9 auf Seite 146). Zusätzlich können von dieser Liste aus zu den einzelnen Variablen, Funktionen und Regeln ebenfalls durch ein spezielles Stylesheet generierte Detailansichten aufgerufen werden. Diese enthalten neben der Dokumentation den jeweiligen Quelltext mit Syntax-Highlighting und verlinkten Cross-Refe-

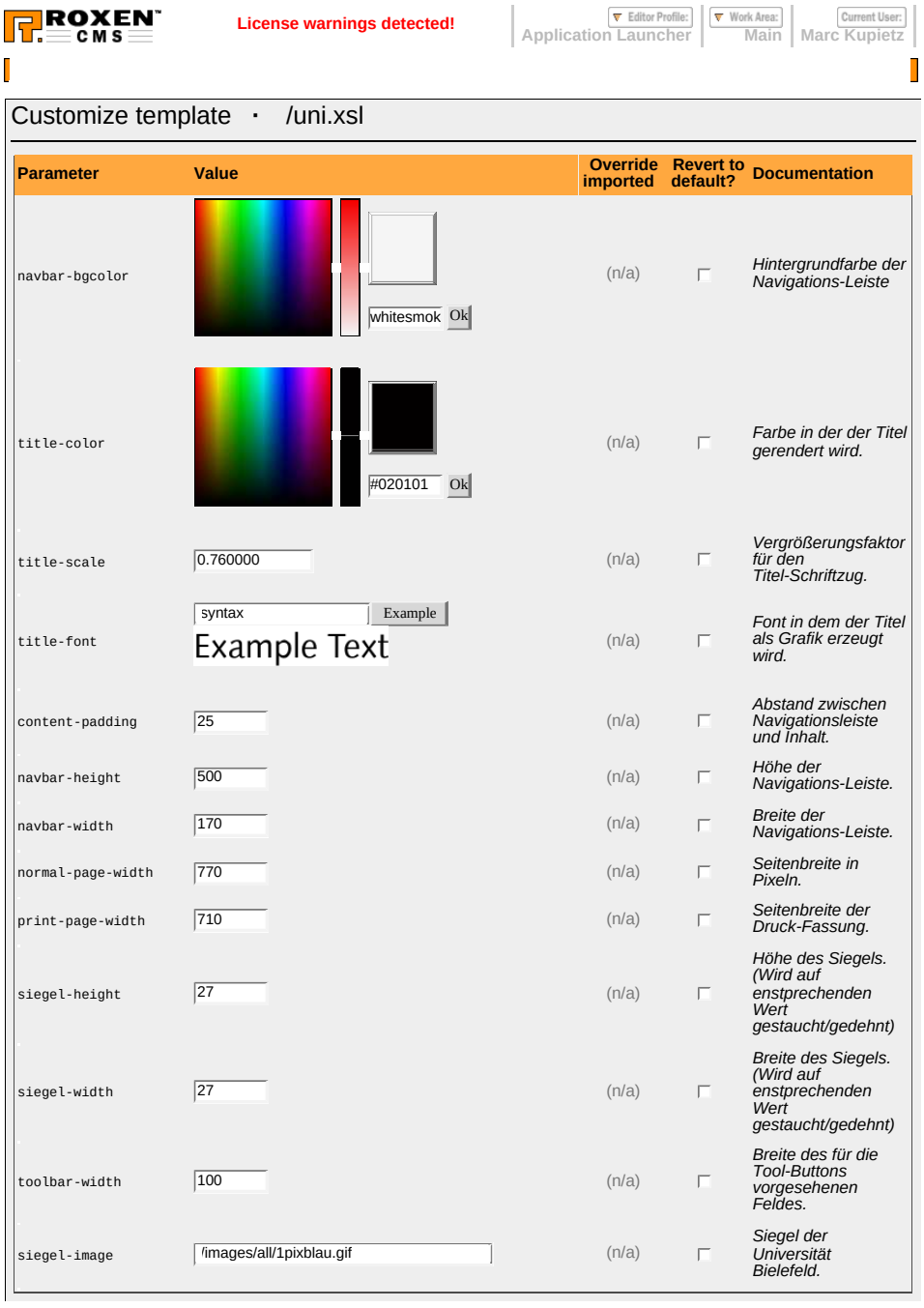


**Abbildung 5.8:** *Roxens Webschnittstelle zur Einstellung von Variablen XSLT-Stylesheets.* Roxen erlaubt nicht nur die Einstellung der Variablen, die im Stylesheets selbst instantiiert werden, sondern auch die Einstellung importierter Variableninstantiierungen.


## Variablen

Im Folgenden sind die "dokumentierten" Variablen des Unibi-Standadttemplates aufgelistet. Diese Variablen werden ihre beschriebene Funktionsweise mit hoher Wahrscheinlichkeit behalten und dürfen von Templates, die das Unibi-Standadttemplate importieren benutzt und überschrieben werden. Wenn Sie den Namen einer Variable anklicken, wird ihre aktuelle Definition im Uni-Standard-Template angezeigt.

| Variable | Beschreibung |
|---|---|
| `normal-page-width` | Seitenbreite in Pixeln. |
| `print-page-width` | Seitenbreite der Druck-Fassung. |
| `navbar-width` | Breite der Navigations-Leiste. |
| `navbar-height` | Höhe der Navigations-Leiste. |
| `navbar-bgcolor` | Hintergrundfarbe der Navigations-Leiste |
| `content-padding` | Abstand zwischen Navigationsleiste und Inhalt. |
| `toc-entries` | Legt fest, welche Elemente ins Inhaltsverzeichnis der Seite eingetragen werden. Ein Wert von "//h2" würde z.B. nur alle h2-Überschriften eintragen. |
| `title` | Titel der im Seitenkopf per <gtext> angezeigt wird. |
| `title-font` | Font in dem der Titel als Grafik erzeugt wird. |
| `title-color` | Farbe in der der Titel gerendert wird. |
| `title-scale` | Vergrößerungsfaktor für den Titel-Schriftzug. |
| `title-bgtext` | Hintergrund-Schriftzug des Titels. |
| `title-bgimage` | Hintergrund-Bild des Titels. |
| `window-title` | Titel der Schiebe-Leiste des Browser-Fensters angezeigt wird. |
| | Logo ganz links oben, über der Navigations-Leiste. Wird |

**Abbildung 5.9:** *Dokumentierte Variablen und Funktionen des Uni-Standard-Stylesheets.* Ausschnitt aus der teilweise automatisch erzeugten Dokumentationsseite.


**Abbildung 5.10:** Automatisch generierte Dokumentation zur Funktion unibi:get-title

renzen (Abbildung 5.10 ).

## 5.6 Semantic-Web-Anwendungen und Syndizierung

### 5.6.1 Content-Syndication

Auch im Hochschulbereich wird eine – häufig mit dem Ausdruck *Content Syndication* bezeichnete – Verteilung von Inhalten zu deren Weiterverarbeitung oder Einbindung in externe Systeme immer wichtiger. Grundsätzlich muss dabei zwischen einem unspezifischen Angebot weiterverarbeitbarer Informationen und der zur Verfügungstellung von für einen bestimmten Verwender oder Verwendungszweck vorgesehener Informationen differenziert werden.

An der Universität Bielefeld besteht insbesondere Bedarf am Austausch von Inhalten zwischen dem zentralem CMS, dem Webserver der Bibliothek, verschiedenen E-Learning-Systemen, dem *Bielefelder Informationssystem* (BIS), zukünftigen Systemen zur Verwaltung von Dokumenten (Verwaltung) und Bildern (Pressestelle), sowie den Web-Services verschiedener Einrichtungen. Der Grund für die Notwendigkeit eines solchen hochschulinternen Austauschs besteht zum einen in der Notwendigkeit der Existenz der verschiedenen Systeme. Diese ist – ganz abgesehen von der Unmöglichkeit eines von ihnen abzuschaffen – vor allem dadurch gegeben, dass jedes der genannten Systeme – *die jeweilig zugehörige personellen Infrastruktur eingeschlossen* – über spezielle Fähigkeiten verfügt, die durch die jeweilig anderen Systeme nicht geleistet werden können. Zum anderen ist es aber notwendig, die jeweilig zur Verfügung gestellten Funktionen und Informationen in verschiedenen Kontexten nutzen zu können, ohne dabei durch Kopien schwer handhabbare Redundanzen zu erzeugen. Inhalte von Lehrveranstaltungen sollten z. B. im auf die Verwaltung und Aufbereitung Content spezialisierten CMS gepflegt und dargestellt werden können, aber auch – komplett oder teilweise – in speziellen E-Learning-Systemen in einem evtl. um andere Lehrtexte oder Übungen erweiterten didaktischen Kontext nutzbar sein. Außerdem müssen natürlich alle vorhandenen Inhalte und zugehörige Metadaten für die Integration in den zentralen Webauftritt und potentiell auch für die Integration in externe Sammlungen verfügbar sein.

Voraussetzung für einen solchen *Austausch von Inhalten* über eine reine Verlinkung hinaus ist neben der Einigung auf Kommunikationsprotokolle[19] eine Einigung auf

[19]Die Kommunikationsprotokolle können unter Umständen ebenso wie genauere Spezifikationen der angebotenen Dienste durch die *Web Services Description Language* (WSDL)

Inhaltsformate bzw. Auszeichnungssprachen. Für den hochschulinternen Gebrauch kommen als Austauschformat – eine flexible Konfigurierbarkeit oder Programmierbarkeit der beteiligten Systeme vorausgesetzt – inhaltsspezifische XML-Sprachen in Frage, da diese leicht auf ihre syntaktische Korrektheit überprüfbar sind und die Synchronisation zwischen intendierter und interpretierter Semantik innerhalb eines solch engen Rahmens kontrollierbar ist. Für die Syndizierung von Inhalten für externe Anwendung bietet sich dagegen die Verwendung von RDF-XML als Basisformalismus an, auf Basis dessen – wie in Kapitel 3 ausführlich dargelegt – eine maximale semantische Interoperabilität erzielt werden kann.

In Verbindung mit dem CMS der Universität Bielefeld wurde bislang vor allem eine Einbindung von Inhalten des *Bielefelder Informationssystems* (BIS) realisiert. Das System selbst basiert auf einem Java-Applikationsserver und verwaltet u.A. das elektronische kommentierte Vorlesungsverzeichnis (eKVV), Veranstaltungshinweise, aktuelle Meldungen und ein Personal- und Telefonverzeichnis der Universität und stellt diese Daten im Intranet und z.T. im Extranet zur Verfügung. Eingebunden in den zentralen Webauftritt der Universität werden vor allem Veranstaltungshinweise, aktuelle Meldungen und Pressemitteilungen. Die inhaltliche Auswahl und Aufbereitung wird dabei vom BIS-System und die Aufbereitung für die externe Präsentation vom CMS geleistet (siehe Abschnitt 5.6.3 zu Implementationsdetails). Die Präsentation der aktuellen Meldungen erfolgt dabei nicht nur in HTML-Form zur Anzeige im Webbrowser, sondern auch in Form einer *RDF Site Summary* (RSS) (siehe Abschnitt 3.3.4.2), um die Meldungen auch für andere Sites und für Newstickerapplikationen verfügbar zu machen.[20]

## 5.6.2 Metadatenexport im RDF/XML-Format

Eine weitere in den Universitätsseiten implementierte RDF-Anwendung ist der Export von Metadaten im RDF/XML-Format zu jeder Seite, die das Standard-Stylesheet oder eine Spezialisierung dieses verwendet (siehe Beispiel 5.6.1 ). Standardmäßig werden dabei nur basale Dublin-Core- und VCard-Informationen[21] exportiert, da aufgrund Roxens eingeschränkten Metadateninventars normalerweise keine darüber hinausgehenden Informationen vorhanden sind. Durch Stylesheetspezialisie-

[siehe WSD03; CZD+01] festgelegt werden.

[20] Die *Aktuellen Meldungen* der Universität Bielefeld können unter `http://www.uni-bielefeld.de/Universitaet/Aktuelles/meldungen.rdf` im RDF/RSS-Format abgerufen werden.

[21] siehe Abschnitt 3.3.4.2 bzw. [KS02]

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE rdf:RDF
  PUBLIC "-//DUBLIN CORE//DCMES DTD 2002/07/31//EN"
  "http://dublincore.org/documents/2002/07/31/dcmes-xml/dcmes-xml-dtd.dtd">
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
    rdf:about="http://www.uni-bielefeld.de/lili/kommissionen.html">
    <dc:title>Kommissionen</dc:title>
    <dc:creator>Marc Kupietz</dc:creator>
    <dc:description>
      Kommission für Forschung und wissenschaftlichen Nachwuchs;
      Kommission für Lehre und studentische Angelegenheiten;
      Kommission für Struktur, Planung und Personalangelegenheiten;
      Magisterprüfungsausschuss; Promotionsausschuss;
      Fraüngleichstellungskommission; Bibliothekskommission
    </dc:description>
    <dc:subject>Kommissionen</dc:subject>
    <dc:publisher>
      Universität Bielefeld, Fakultät für Linguistik und
      Literaturwissenschaften
    </dc:publisher>
    <dc:date>2003-02-11</dc:date>
    <dc:language>de</dc:language>
    <dc:format>text/html</dc:format>
  </rdf:Description>
</rdf:RDF>
```

**Beispiel 5.6.1:** Automatisch generierte Dublin-Core-Metadaten im RDF/XML-Format

rungen ist es jedoch möglich, die exportierten Metadaten beliebig – abhängig von der jeweiligen Ressource – dynamisch zu erweitern. Dieses Feature wird zum einen durch Roxens dynamische Anwendbarkeit von Stylesheets auf Ressourcen und zum anderen durch die spezielle Art der Implementation erreicht, die im folgenden kurz skizziert werden soll.

An der Generierung von RDF/XML-Metadaten sind normalerweise zwei Stylesheets beteiligt: Das auch für die HTML-Aufbereitung der jeweiligen Ressource verantwortliche Stylesheet generiert[22] im Kopf der Ausgabe einen Link vom Relationstyp *meta* auf die zugehörige RDF-Beschreibung:

[22] entsprechend der in [KS02] empfohlenen Vorgehensweise

```
<DC:Creator rdf:parseType="Resource">
  <vCard:FN>Marc Kupietz</vCard:FN>
  <vCard:N rdf:parseType="Resource">
    <vCard:Family>Kupietz</vCard:Family>
    <vCard:Given>Marc</vCard:Given>
  </vCard:N>
  <vCard:EMAIL rdf:parseType="Resource">
    <rdf:value>marc.kupietz@uni-bielefeld.de</rdf:value>
    <rdf:type rdf:resource="http://www.w3.org/2001/vcard-rdf/3.0#internet"/>
  </vCard:EMAIL>
</DC:Creator>
```

**Beispiel 5.6.2:** *Angaben zum Autor im RDF/XML-vCard-Format.*
Mögliche Erweiterung der in Beispiel 5.6.1 dargestellten RDF-Beschreibung.

```
<link href="/Universitaet/index.html?__xsl=/templates/rdf.xsl"
      rel="meta"/>
```

Bei der RDF-Beschreibung handelt es sich natürlich nicht um eine statische Datei, sondern um die Ausgabe des zweiten Stylesheets `rdf.xsl` angewendet auf die aktuelle Ressource (hier: `/Universitaet/index.html`). Das Stylesheet `rdf.xsl`, das aufgrund der Roxen-spezifischen CGI-Variable `__xsl` angewendet wird, erzeugt – abhängig von Ressource und bestimmten Variablen – DC-Metadaten im RDF/XML-Format. Außerdem importiert es wiederum das erste Stylesheet und ruft dieses mit dem Inhalt der Ressource im »Metadatenmodus« auf:

```
<xsl:apply-templates mode="metadata"/>
```

So können, im ansonsten für die grafische Aufbereitung verwendeten Stylesheet, Regeln (mit dem Modus `metadata`) definiert werden, die, abhängig vom Inhalt der Ressource oder anderen Bedingungen, weitere RDF-Metadaten generieren. Denkbar sind z.B. Verweise auf andere Ressourcen-Varianten (ältere/ neuere Versionen oder Versionen in anderen XML-Sprachen) oder die genauere Spezifizierung von Beschreibungsobjekten[23] durch die Angabe beschriebener, (anonymer) Ressourcen an Stelle von Literalen (siehe Beispiel 5.6.2).

Um *Metadaten* nicht auf eine *Beschreibung der aktuellen Ressource* einzuschränken, sondern auch die Beschreibung beliebiger anderer Ressourcen zuzulassen, ruft das Standard-RDF-Stylesheet außerhalb des Elements zur Beschreibung der aktuellen

[23] *Objekt* ist hier im Sinne der RDF-Terminologie zu verstehen – siehe Abschnitt 3.3.1 bzw. Abbildung 3.1 auf Seite 62

Ressource wiederum importierte Regeln, diesmal im »RDF-Modus« auf:

```
<xsl:apply-templates mode="rdf"/>
```

Dies ermöglicht insbesondere die Generierung von RDF-(Meta-)Daten, die sich nicht auf die aktuelle Ressource, sondern auf ihren *Gegenstand*, bzw. *Inhalt* beziehen. Auf diese Art und Weise können Spezialisierungen des Standard-Stylesheets Metadaten und Inhalte nicht nur für die Ansicht im Browser, sondern auch für die Weiterverarbeitung durch andere Anwendungen, also – wie in Abschnitt 3.3.5 bereits skizziert – nicht nur für das *Web*, sondern auch für das *Semantic Web* aufbereiten und damit auch so etwas wie ein redundanzfreies »*Single Source – Cross Application Publishing*« erreichen.

Innerhalb des Universitätsszenarios ist die wohl wichtigste zukünftige Anwendung solcher »semantischer« Aufbereitungen die Verfügbarmachung von Lernmodulen für verschiedene Lernkontexte und Lernplattformen. Zur Zeit, jedoch, scheitert dies zum einen daran, dass die heute maßgeblichen Standardformate für E-Learning-Metadaten wie IMS[24] und IEEE LOM[25] nicht auf RDF(S), sondern direkt auf XML basieren. Zwar können natürlich mit Hilfe der oben dargestellten dynamischen XSLT-Transformationen nicht nur RDF/XML-, sondern auch z. B. IMS/XML-konforme Metadaten zu entsprechend annotierten Lerninhalten generiert werden, eine wirklich *distribuierte* (Wieder-)Verwendbarkeit dieser ist jedoch wegen der in Kapitel 3 genannten Gründe sehr unwahrscheinlich.[26] Dementsprechend sehen zumindest kommerzielle E-Learning-Plattformen in der Regel auch keine dynamische Nutzung von Metadaten und keine dynamische Inkorporierung von Lerninhalten vor.

### 5.6.3 Implementation der Einbindung syndizierten Contents in das CMS

Wie bereits im Zusammenhang mit den Charakteristika des Roxen CMS in Abschnitt 5.2.3.1 erwähnt, stellt die Scriptsprache RXML Möglichkeiten zur Verfügung, Daten von anderen HTTP-Servern zu verarbeiten und in eigene Seiten einzubinden. Da jedoch die entsprechende RXML-Anweisung <`insert href="`*`URI`*`"/`> das CMS während des Wartens auf die Antwort vom externen HTTP-Servers blockiert, wird – um

[24] IMS Global Learning Consortium (`http://www.imsglobal.org`)
[25] IEEE Learning Object Metadata (`http://ltsc.ieee.org/wg12/`)
[26] vgl. [NPN02] zum Thema XML vs. RDF zur Repräsentation von E-Learning-Metadaten

die Funktionalität des CMS von externen Systemen unabhängig zu halten – zur Einbindung externer Ressourcen ein anderes Verfahren verwendet:
Das Standard-Stylesheet transformiert das `insert`-Element in zwei RXML-Datenbankzugriffs-Ausdrücke. Der erste Ausdruck schreibt den URI der angeforderten Ressource in eine Art *Anfragetabelle*. Der zweite holt die Ressource – falls diese bereits vorhanden ist – aus einer Art *Puffertabelle*. Für das Abarbeiten der *Anfragetabelle* bzw. das Auffüllen und Auffrischen der *Puffertabelle* sorgt ein separates, permanent aktives Perl-Script. Um das Ergebnis der Datenbankanfrage (also die externe Ressource) – das normalerweise erst im zweiten Transformationsschritt erzeugt würde[27] – mit in die XSLT-Transformation einzubeziehen, kann die Verarbeitung der Datenbankanfrage optional durch die Einbettung in das Roxen-spezifische XSLT-Erweiterungselement `rxml:parse` vorgezogen werden. Unabhängig davon besteht auch die Möglichkeit, die externe Ressource zunächst mit separaten XSLT-Stylesheets zu transformieren. Auch eine Überführung von HTML in wohlgeformtes XML kann optional vorgeschaltet werden, was die Unabhängigkeit von externen Systemen und die Menge inkorporierbarer Inhalte weiter erhöht.

[27] siehe Abschnitt 5.5

# 6 Zusammenfassung und Ausblick

## 6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde am Beispiel der Universität Bielefeld demonstriert, wie sich eine effiziente, funktionale und entwicklungsfähige Internetpräsenz einer Hochschule realisieren lässt.

Im ersten Kapitel wurden zunächst die für eine syntaktisch interoperable Repräsentation von Inhalten relevanten Formalismen (insbesondere XML und angrenzende Standards) eingeführt. Im Hinblick auf eine möglichst uneingeschränkte Wiederverwendbarkeit und Nachhaltigkeit von Inhalten ist die Verwendung von XML allein allerdings keine hinreichende Voraussetzung. Vielmehr müssen XML-Auszeichnungssprachen auch im *generischem* Sinne verwendet werden.

Weiterhin wurden im Rahmen eines Vergleichs mit konventionellen SQL-Datenbanksystemen die jeweiligen primären Anwendungsfelder zunächst theoretisch eingegrenzt und im Hinblick auf die jeweiligen Schnittstellen zur Abfrage und Manipulation von Daten diskutiert. Dabei zeigte sich unter anderem, dass XML-Systeme besonders bei der Datenmanipulation noch nicht den Grad der (Inter-)Operabilität von Datenbanksystemen erreicht haben.

Ein weiteres Problem im Zusammenhang mit der Anwendung von XML-Systemen wurde im folgenden Abschnitt zum Thema XML-Benutzerschnittstellen diskutiert: *Generisches Markup* ist zur Zeit weder konzeptuell noch innerhalb von Mainstream-Softwareapplikationen etabliert. Insbesondere stößt die Umsetzung einer Entkoppelung von Darstellung und Inhalt in der Praxis häufig auf fehlende Akzeptanz. Ursachen dafür sind im Wesentlichen zu komplexe Dokumentgrammatiken, unzureichendes Verständnis von *generischem Markup* und schlechte Softwareunterstützung. Jedoch werden sich diese Ursachen durch die Verbreitung von XML-Schema, Schulung und spezielle XML-Editoren bzw. XML-basierte Webformulare (XForms) vermutlich ausräumen lassen.

Im zweiten Kapitel der Arbeit wurden Stylesheetsprachen und Programmiertech-

niken vorgestellt, mit deren Hilfe sich *generische*, also auf die Art ihres Inhalts spezialisierte, Auszeichnungssprachen in Sprachen mit definierter, hauptsächlich darstellungsorientierter Semantik (HTML und *XSL Formatting Objects*) übersetzen lassen. Als grundlegender Formalismus ist dazu die *eXtensible Stylesheet Language: Transformations* (XSLT) aufgrund ihrer deklarativen und funktionalen Eigenschaften und ihrer auf Baumtransformationen spezialisierten Möglichkeiten am besten geeignet. Zur Aufbereitung von dokumentexternen Datenquellen werden allerdings zusätzlich konventionelle Script- oder Programmiersprachen benötigt. Als Ergebnis des zweiten Kapitels wurde deshalb eine mehrstufige Architektur entworfen, mit Hilfe derer sich (unter Verwendung von XSLT, eingebetteten Scriptsprachen, CSS und XSL-FO) Dokumente und Daten integrieren und im Sinne eines *Single Source Publishing* aufbereiten lassen.

In den ersten beiden Kapiteln wurde das Thema *Semantik* im Wesentlichen im Zusammenhang mit dokumentklassenspezifischen bzw. darstellungsorientierten Auszeichnungssprachen behandelt, deren Bedeutung letztlich allein durch spezialisierte, verarbeitende Anwendungen oder den menschlichen Betrachter konstituiert werden. Thema des dritten Kapitel ist dagegen, wie sich auf der Grundlage des *Resource Description Frameworks* (RDF) *automatisch interpretierbare*, erweiterbare (Metadaten-) Sprachen und ein sogenanntes *Semantic Web* entwickeln können. Solche *semantisch interoperablen* (Metadaten-)Sprachen sind notwendig, um die *Nachhaltigkeit* und *Wiederverwendbarkeit* von Inhalten insbesondere im Hinblick auf ihre Wiederauffinbarkeit garantieren zu können. Zusätzlich werden sie aber auch benötigt, um in Zukunft verschiedene Web-Services und Office-Applikationen besser und flexibler verketten zu können.

Im Rahmen der Diskussion von RDF und RDF-Schema wurde dafür argumentiert, dass obwohl sich die Semantik von RDF(S)-Anwendungen nicht vollständig formal festlegen lässt und selbst die Semantik von RDF-Schema nicht vollständig formalisiert ist, zumindest die Entstehung einer Art »*Pragmatic Web*« durchaus wahrscheinlich ist. Auf der einen Seite wird dazu – ähnlich wie bei der Entwicklung natürlicher Sprachen – die Bildung von Konventionen innerhalb von Interessengemeinschaften beitragen. Auf der anderen Seite lässt sich auch das *Semantic Web* auf Grundlage der in den ersten beiden Kapiteln dargestellten Auszeichnungs- und Transformationsstrategien *automatisch* mit Inhalten populieren. So kann auch das *Semantic Web* die »kritische Masse« erreichen, die für ein auf der Eigendynamik des Netzes beruhendes weiteres Wachstum notwendig ist.

Kapitel 4 widmete sich der Frage, wie sich die zuvor dargestellten Repräsenation-

formalismen und Aufbereitungstechniken in integrierten Softwaresystemen, sogenannten *Content Management Systemen* (CMS), implementieren lassen und zweitens, wie sich innerhalb solcher Systeme die im Rahmen der Arbeit aufgestellten Prinzipien (insbesondere Redundanzvermeidung und Interoperabilität) in Bezug auf die *Organisation* und *Verwaltung* von Inhalten verwirklichen lassen.

Eine grundsätzliche Anforderung an Softwaresysteme, die die Speicherung, Organisation, Aufbereitung und Auslieferung von Inhalten integrieren, ist zunächst die Implementation von Standardtechnologien, um Abhängigkeiten zu vermeiden und die Zugänglichkeit und Nachhaltigkeit eingestellter Inhalte zu gewährleisten.

Im Hinblick auf die Organisation von Inhalten ist insbesondere die Verwaltung von verschiedenen Kategorien von Ressourcen-Versionen (»*diachrone*«, »*synchrone*«, öffentliche/ nicht-öffentliche) und -Varianten (Sprachvarianten) von Bedeutung. In diesem Zusammenhang wurden Techniken skizziert, mit Hilfe derer sich ein solches Versions- und Variantenmanagement in einem CMS realisieren lässt.

Hinsichtlich der Aufbereitung von Inhalten müssen insbesondere Möglichkeiten zum Zugriff auf alle innerhalb solcher integrierter Systeme vorhandenen Daten (z. B. Verzeichnishierarchie, Metadaten, Grafiken, ...) zur Verfügung stehen. Entsprechend der zusätzlichen Anforderungen, wurde die in Kapitel 2 skizzierte Aufbereitungsarchitektur erweitert. Grundlage dieser Erweiterung bildet eine *uniforme* Repräsentation bzw. Verarbeitbarkeit aller im System vorhandener Daten bzw. Informationen. Zur performanten Realisierung der Aufbereitung wurde außerdem ein Caching-Mechanismus (»*Teilbaum-Caching*«) sowie ein Verfahren zur Reduktion von Aufbereitungsabhängigkeiten (»*Parameter-Mapping*«) vorgeschlagen.

In Kapitel 5 wurde schließlich der Relaunch des Webauftritts der Universität Bielefeld dokumentiert und dabei demonstriert, wie sich die in den ersten Kapiteln aufgestellten Prinzipien und dargestellten Techniken im Kontext einer Hochschule verwirklichen lassen.

Ausgangspunkt der Darstellung bildete der alte Webauftritt der Universität, der vor allem wegen fehlender Möglichkeiten zur Entkoppelung von Inhalt, Präsentation (und Site-Navigation) und den daraus resultierenden Problemen (schlechte Wartbarkeit und Funktionalität, uneinheitliches Erscheinungsbild, ...) eine vollständige Neu-Konzeption notwendig machte. Grundlage dazu bildeten zum einen personelle Maßnahmen und zum anderen die Anschaffung des CMS von der Firma Roxen, welches die in Kapitel 4 aufgestellten Anforderungen zum Teil erfüllte. Die diesbezüglich wichtigste Einschränkung des Roxen-CMS ist jedoch die fehlende Möglichkeit zur Erweiterung von Metadaten.

Die Vorbereitungsarbeiten für den Relaunch umfassten im Einzelnen: den Entwurf eines tragfähigen Layouts, die Neu-Strukturierung von Inhalten, die Entwicklung eines entsprechenden Konzept zur Navigation und die Implementation von CMS, Stylesheets und Scripts. Außerdem wurde eine Konzept zur Eigendarstellung (»*Corporate Sub-Identities*«) und zur Migration für die Fakultäten und Einrichtungen der Universität entworfen. Kern der Migrationsstrategie war, eine sukzessive Ausschöpfung neuer Möglichkeiten anzubieten.

Im Rahmen der Vorstellung des neuen Webauftritts wurde dann zunächst das aus den hochschulspezifischen Anforderungen sowie der automatisierten Generierung von Navigationselementen resultierende Konzept einer flexiblen, »parametertoleranten« Gestaltung erläutert. Im Folgenden wurden dann die auf Grundlage von Metadaten und einer redundanzvermeidenden Ausnutzung der Dateisystemhierarchie automatisch generierten Hauptnavigationselemente im Einzelnen beschrieben.

Die Implementation des neuen Webauftritts basiert auf der in Kapitel 2 und 4 entwickelten mehrstufigen Architektur zur Integration und Aufbereitung von Inhalten. Auf Grundlage dieser Architektur konnte außerdem – wie in Kapitel 3 in Aussicht gestellt – eine automatische Generierung von RDF/XML-Aussagen zur Syndizierung von Content aus XML-Dokumenten und Metadaten realisiert werden. Ein weiterer wichtiger Aspekt der Implementation war die Umsetzung von *Corporate Sub-Identities* durch eine Hierarchie von XSLT-Stylesheets. Die zur dezentralen Entwicklung und Wartung dieser *Sub-Identities* benötigte Dokumentation wird dabei, ebenfalls auf Grundlage der allgemeinen Aufbereitungsarchitektur, redundanzvermeidend, *on-the-fly* generiert.

## 6.2 Ausblick

Noch nie wurde so viel Information produziert wie heute. Noch nie ist allerdings auch so viel Information verloren gegangen. Die Sicherung der *Nachhaltigkeit und Wiederverwendbarkeit* von Inhalten wird deshalb in Zukunft von zentraler Bedeutung sein.

Die *Erhaltung von Wissen* ist dabei jedoch nicht allein im Hinblick auf eine Konservierung für kommende Generationen zu verstehen. Zunächst geht es vielmehr darum, Wissen überhaupt so repräsentieren zu können, dass nicht bereits bei seiner elektronischen Erfassung relevante Informationen verloren gehen. In Bezug auf Texte und Daten ist dabei die Möglichkeit, diese strukturieren zu können sowie

Elemente der Struktur mit generischen Zusatzinformationen versehen zu können, von fundamentaler Wichtigkeit. XML und zugehörige Standards legen bereits heute die dafür notwendigen Voraussetzungen. Angesichts des Ziels einer Informationserhaltung, wird es aber in Zukunft immer wichtiger werden, auch die *Semantik* von Struktur und Auszeichnungen innerhalb des XML-Frameworks und unabhängig von (vergleichsweise »vergänglichen«) Verarbeitungsanwendungen formal festlegen zu können. Das durch XML-Schema (XSD) eingeführte hierarchische Typenkonzept könnte dabei eine wichtige Rolle spielen. Jedoch müssen sich die durch die Typenhierarchie herstellbaren Relationen zwischen Element- und Attributtypen auch in Bezug auf ihre Bedeutung, insbesondere durch zukünftige XSLT/XPath-Versionen, ausnutzen lassen.

Hinsichtlich eines »physikalischen« Erhalts von Inhalten auf Ressourcenebene, wird die Entwicklung weniger von neuen Standards, sondern hauptsächlich von der Software, bzw. von *Content Management Systemen* abhängen. Entscheidend wird dabei sein, inwieweit diese eine in Kapitel 4 dieser Arbeit skizzierte Organisation diachroner Versionen von Ressourcen unterstützen werden. In Bezug auf die Interoperabilität solcher Lösungen ist dabei natürlich auch auf die Implementation entsprechender Standardprotokolle sowie die Bildung von Metadatenkonventionen zur Repräsentation von Relationen zwischen Ressourcen (z. B. *superceeds*, *is-superceeded-by*) von Bedeutung.

Wie bereits ausführlich dargelegt wurde, werden *Metadaten* generell eine zentrale Rolle im Hinblick auf die Gewährleistung der Nachhaltigkeit von elektronisch gespeichertem Wissen einnehmen. Um sicherzustellen, dass Inhalte im *World Wide Web* nicht verloren gehen, müssen diese mit Metadaten ausgezeichnet sein. Um die »Last der Suche« außerdem auf Softwareagenten verlagern zu können, müssen diese Metadaten erstens automatisch interpretierbar sein und zweitens Ressourcen über semantisch transparente, eindeutige Relationen miteinander vernetzen können. Das *Resource Description Framework* (RDF) legt dafür eine geeignete Grundlage.

In der bisherigen Diskussion wurde *Nachhaltigkeit und Wiederverwendbarkeit* im Wesentlichen vor dem Hintergrund zukünftiger Standards und deren Implementation in Softwaresystemen betrachtet. Viel entscheidender für eine Entwicklung in Richtung des avisierten Ziels ist jedoch, dass sich zunächst zumindest die elementaren Voraussetzungen, wie insbesondere die *generische Auszeichnung von Inhalten*, auch *konzeptuell* etablieren. Dies ist keineswegs selbstverständlich und setzt z.T. erhebliche Umdenkprozesse voraus.

Diese Umdenkprozesse sind vor allem natürlich da wichtig, wo viel erhaltungs-

würdiges Wissen produziert und vermittelt wird, also ganz besonders an Universitäten. Wenn sie nicht stattfinden, droht sich gerade im aufgrund veränderter Lerngewohnheiten immer mehr an Bedeutung gewinnenden Bereich der elektronischen Vermittlung von Wissen (*E-Learning*) eine ähnliche Krise zu wiederholen, wie sie bereits die Softwareindustrie in den 60er- und 70er- und die Computer-Linguistik in den 90er-Jahren, bedingt jeweils durch die schlechte Wiederverwendbarkeit/ Nachhaltigkeit von Software bzw. Lexika, durchgemacht haben.

Nicht zuletzt auch aus ökonomischen Gründen wird es deshalb notwendig sein, dass gerade an Universitäten Inhalte (Forschungsergebnisse, Grundlagenwissen, ...) von vorn herein so repräsentiert werden, dass nicht nur ihre Nachhaltigkeit, sondern auch ihre *Wiederverwendbarkeit in verschiedenen Kontexten* (Publikation, WWW, Präsenzlehre, E-Learning, ...) gewährleistet ist.

# Literaturverzeichnis


[AAB+02] AHMED, Kal; AYERS, Danny; BIRBECK, Mark; COUSINS, John; DODDS, David; LUBELL, Joshua; MILOSLAV, Nick; RIVERS-MOORE, Daniel; WATT, Andrew & WRIGHTSON, Ann: *Professional XML Meta Data*. Birmingham, UK: Wrox Press, 2002

[ABS00] ABITEBOUL, Serge; BUNEMANN, Peter & SUCIU, Dan: *Data on the Web: From Relations to Semistructured Data and XML*. San Francisco, CA: Kaufmann, 2000

[Alv95] ALVESTRAND, H.: *Tags for the Identification of Languages*. IETF. 3/1995. RFC 1766. `http://www.ietf.org/rfc/rfc1766.txt`

[Bax02] BAX, Ingo: *Entwicklung eines webbasierten Softwaresystems zur Transkription, Annotation und Analyse von Sprachdaten*, Universität Bielefeld, Technische Fakultät, Unveröffentlichte Diplomarbeit, 2002

[BL98a] BERNERS-LEE, Tim: *Cool URIs don't change*. W3C. 1998. W3C Article. `http://www.w3.org/Provider/Style/URI.html`

[BL98b] BERNERS-LEE, Tim: *What the Semantic Web can represent*. W3C. 9/1998. W3C Design Issues. `http://www.w3.org/DesignIssues/RDFnot.html`

[BLFF96] BERNERS-LEE, T.; FIELDING, R. & FRYSTYK, H.: *Hypertext Transfer Protocol – HTTP/1.0*. IETF. 5/1996. RFC 1945. `http://www.ietf.org/rfc/rfc1945.txt`

[BLFM98] BERNERS-LEE, T.; FIELDING, R. & MASINTER, L.: *Uniform Resource Identifiers (URI): Generic Syntax*. 8/1998. RFC 2396. `http://www.ietf.org/rfc/rfc2396.txt`

[Bou02] BOURRET, Ronald: *XML and Databases*. 1999-2002. – Forschungsbericht. http://www.rpbourret.com/xml/XMLAndDatabases.htm

[Bra01] BRAY, Tim: *What is RDF?* XML.com, O'Reilly, 24.01.2001. – http://www.xml.com/pub/a/2001/01/24/rdf.html

[CAE+02] CLEMM, G.; AMSDEN, J.; ELLISON, T.; KALER, C.; WHITEHEAD, J. & CRUZ, U.C. S.: *Versioning Extensions to WebDAV*. IETF. 3/2002. RFC 3253. http://www.ietf.org/rfc/rfc3253.txt

[Cov98] COVER, Robin: XML and Semantic Transparency. *The XML Cover Pages*. 11/1998. – http://xml.coverpages.org/xml/xmlAndSemantics.html

[Cov00] COVER, Robin: The SGML/XML Aversion to Semantics. *The XML Cover Pages*. 09/2000. – http://xml.coverpages.org/xml/sgmlEschewSemantics.html

[CRF00] CHAMBERLIN, Don; ROBIE, Jonathan & FLORESCU, Daniela: Quilt: an XML Query Language for Heterogeneous Data Sources. In: *Lecture Notes in Computer Science*. Heidelberg: Springer-Verlag, 12/2000

[CSS98] LIE, Håkon W.; BOS, Bert; LILLEY, Chris & JACOBS, Ian (Hrsg.): *Cascading Style Sheets, level 2*. W3C. 12.5.1998. CSS2 Specification. http://www.w3.org/TR/REC-CSS2/

[CZD+01] CHOPRA, Vivek; ZORAN, Zaev; DAMSCHEN, Gary; DIX, Chris; CAULDWELL, Patrick; CHAWLA, Rajesh; SAUNDERS, Kristy; OLANDER, Glenn; NORTON, Francis; HONG, Tony; OGBUJI, Uche & RICHMAN, Mark A.: *Professional XML Web Services*. Birmingham, UK: Wrox Press, 2001

[Dat87] DATE, C.J.: *A guide to the SQL standard : a user's guide to the standard relational language*. Reading, MA: Addison Wesley, 1987

[DOM00] HORS, Arnaud L.; HÉGARET, Philippe L.; WOOD, Lauren; NICOL, Gavin; ROBIE, Jonathan; CHAMPION, Mike & BYRNE, Steve (Hrsg.): *Document Object Model (DOM) Level 2 Core Specification*. W3C. 13.11.2000. W3C Recommendation. http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/

[Don66] DONNELLAN, Keith: Reference and Definite Descriptions. *Philosophical Review* 75 (1966), S. 281–304

[DS98] DAWSON, F. & STENERSON, D.: *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. IETF. 11/1998. RFC 2445. http://www.ietf.org/rfc/rfc2445.txt

[Dum00] DUMBILL, Edd: *Distributed XML*. XML.com, 06.09.2000. – http://www.xml.com/lpt/a/2000/09/06/distributed.html

[FGM+99] FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEECH, P. & BERNERS-LEE, T.: *Hypertext Transfer Protocol – HTTP/1.1*. IETF. 6/1999. RFC 2616. http://www.ietf.org/rfc/rfc2616.txt

[Ges03] GESEMANN, Michael: Manipulation von XML-Dokumenten in Tamino. In: WEIKUM, Gerhard; SCHÖNING, Harald & RAHM, Erhard (Hrsg.): *BTW 2003: Datenbanksysteme für Business, Technologie und Web*. Bonn: Köllen Verlag, 2003

[Gre01] GREENBLATT, Bruce: *Internet Directories : How to build and manage applications for LDAP, DNS, and other directories*. Upper Saddle River, NJ: Prentice Hall, 2001

[GWF+99] GOLAND, Y.; WHITEHEAD, E.; FAIZI, A.; CARTER, S. & JENSEN, D.: *HTTP Extensions for Distributed Authoring – WEBDAV*. IETF. 2/1999. RFC 2518. http://www.ietf.org/rfc/rfc2518.txt

[Heu97] HEUER, Andreas: *Objektorientierte Datenbanken*. 2., aktualisierte u. erw. Auflage. Bonn: Addison Wesley, 1997

[Hug00] HUGHES, John: Why functional Programming Matters. In: TURNER, D. (Hrsg.): *Research Topics in Functional Programming*. Reading, MA: Addison Wesley, 2000

[Ina01] INANNELLA, Renato: *Representing vCard Objects in RDF/XML*. W3C. 22.01.2001. W3C Submission Note. http://www.w3.org/TR/vcard-rdf

[Kay00] KAY, Michael: *XSLT: Programmer's Reference*. Birmingham, UK: Wrox Press, 2000

[KS02] KOKKELINK, Stefan & SCHWÄNZL, Roland: *Expressing Qualified Dublin Core in RDF / XML*. Dublin Core Metadata Initiative. 14.04.2002. DCMI Proposed Recommendation. http://dublincore.org/documents/dcq-rdf-xml/

[LB96] LIE, Håkon W. & BOS, Bert: *Cascading Style Sheets, level 1*. W3C. 17.12.1996. W3C Recommendation. http://www.w3.org/TR/REC-CSS1-961217

[LB02] LAFON, Yves & BOS, Bert: *Describing and retrieving photos using RDF and HTTP*. W3C. 19.04.2002. W3C Note. http://www.w3.org/TR/daml+oil-reference

[Leh01] LEHTI, Patrick: *Design and Implementation of a Data Manipulation Processor for an XML Query Language*, Technische Universität Darmstadt, Diplomarbeit, August 2001

[Lew69] LEWIS, David K.: *Convention: A Philosophical Study*. Cambridge, MA: Harvard University Press, 1969

[LL95] LANG, Stefan M. & LOCKEMANN, Peter C.: *Datenbankeinsatz*. Heidelberg: Springer-Verlag, 1995

[LM00] LAUX, Andreas & MARTIN, Lars: *XUpdate — XML Update Language*. XML:DB. 14.09.2000. XML:DB Working Draft. http://www.xmldb.org/xupdate/xupdate-wd.html

[LS99] LASSILA, Ora & SWICK, Ralph R.: *Resource Description Framework (RDF) Model and Syntax Specification*. W3C. 22.02.1999. W3C Recommendation. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[MBK+00] MARTIN, Didier; BIRBECK, Mark; KAY, Michael; LOESGEN, Brian; PINNOCK, Jon; LIVINGSTONE, Steven; STARK, Peter; WILLIAMS, Kevin; ANDERSON, Richard; MOHR, Stephen; BALILES, David; PEAT, Bruce & OZU, Nikola: *Professional XML*. Birmingham, UK: Wrox Press, 2000

[MD02] MEALLING, M. & DENENBERG, R.: *Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations*. IETF. 8/2002. RFC 3305. http://www.ietf.org/rfc/rfc3305.txt

[Moa97] MOATS, R.: *URN Syntax*. IETF. 5/1997. RFC 2141. http://www.ietf.org/rfc/rfc2141.txt

[MSC+02] MAHARRY, Dan; SARAN, Rogerio; CAGLE, Kurt; FUSSELL, Mark & LOPEZ, Nalleli: *Early Adopter XQuery*. Birmingham, UK: Wrox Press, 2002

[Nie00] NIELSEN, Jakob: *Erfolg des Einfachen*. München: Markt+Technik, 2000

[Nov01] NOVATCHEV, Dimitre: *The Functional Programming Language XSLT - A proof through examples*. 11/2001. – Forschungsbericht. http://www.topxml.com/xsl/articles/fp/

[NPN02] NILSSON, Mikael; PALMÉR, Matthias & NAEVE, Ambjörn: Semantic Web Metadata for e-Learning - Some Architectural Guidelines. In: *Proceedings of the Eleventh International World Wide Web Conference*. Honolulu, Hawaii: WWW2002, 5/2002. – http://www2002.org/CDROM/alternate/744/

[P3P02] CRANOR, Lorrie; LANGHEINRICH, Marc; MARCHIORI, Massimo; PRESLER-MARSHALL, Martin & REAGLE, Joseph (Hrsg.): *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. W3C. 16.04.2002. W3C Recommendation. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[PR85] POSTEL, J. & REYNOLDS, J.: *File Transfer Protocol (FTP)*. IETF. 10/1985. RFC 959. http://www.ietf.org/rfc/rfc959.txt

[Pur00] PURDY, Gregor N.: *CVS Pocket Reference*. Sebastopol, CA: O'Reilly, 2000

[RDF02a] HAYES, Pat (Hrsg.): *RDF Model Theory*. W3C. 29.04.2002. W3C Working Draft. http://www.w3.org/TR/2002/WD-rdf-mt-20020429/

[RDF02b] BRICKLEY, Dan & GUHA, R. V. (Hrsg.): *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C. 30.04.2002. W3C Working Draft. http://www.w3.org/TR/rdf-schema/

[Sch00] SCHWARTZ, Aaron: *RDF Site Summary (RSS) 1.0*. RSS-DEV Working Group. 06.12.2000. Official Specification. http://web.resource.org/rss/1.0/

[TH01] TATARINOV, Ives & HALEVY, Weld: Updating XML. In: *Proceedings of ACM SIGMOD 2001*. Santa Barbara, CA: Association for Computing Machinery, 5/2001

[Wal02] WALMSLEY, Priscilla: *Definitive XML schema*. Upper Saddle River, NJ: Prentice Hall, 2002

[WHK97] WAHL, M.; HOWES, T. & KILLE, S.: *Lightweight Directory Access Protocol (v3)*. IETF. 12/1997. RFC 2251. http://www.ietf.org/rfc/rfc2251.txt

[Wit01] WITT, Andreas: *Multiple Informationsstrukturierung mit Auszeichnungssprachen: XML-basierte Methoden und deren Nutzen für die Sprachtechnologie*, Fakultät für Linguistik und Literaturwissenschaften der Universität Bielefeld, Dissertation, 2001. – http://archiv.ub.uni-bielefeld.de/disshabi/2002/0007.pdf

[WM02] WIDHALM, Richard & MÜCK, Thomas: *Topic Maps: Semantische Suche im Internet*. Berlin: Springer-Verlag, 2002

[WSD03] CHINNICI, Roberto; GUDGIN, Martin; MOREAU, Jean-Jacques & WEERAWARANA, Sanjiva (Hrsg.): *Web Services Description Language (WSDL) Version 1.2*. W3C. 24.01.2003. W3C Working Draft. http://www.w3.org/TR/2003/WD-wsdl12-20030124

[XFo02] DUBINKO, Micah; DIETL, Josef; KLOTZ, Leigh L.; MERRICK, Roland & RAMAN, T. V. (Hrsg.): *XForms 1.0*. W3C. 18.01.2002. W3C Working Draft. http://www.w3.org/TR/2002/WD-xforms-20020118/

[XHT02] MCCARRON, Shane; AXELSSON, Jonny; EPPERSON, Beth; NAVARRO, Ann & PEMBERTON, Steven (Hrsg.): *XHTML 2.0*. W3C. 18.12.2002. W3C Working Draft. http://www.w3.org/TR/2002/WD-xhtml2-20021218/

[XML99] BRAY, Tim; HOLLANDER, Dave & LAYMAN, Andrew (Hrsg.): *Namespaces in XML*. W3C. 14.01.1999. W3C Recommendation. http://www.w3.org/TR/REC-xml-names/

[XML00] BRAY, Tim; PAOLI, Jean; SPERBERG-MCQUEEN, C.M. & MALER, Eve (Hrsg.): *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C. 06.10.2000. W3C Recommendation. http://www.w3.org/TR/2000/REC-xml-20001006

[XML01a] FALLSIDE, David C. (Hrsg.): *XML Schema Part 0: Primer*. W3C. 02.05.2001. W3C Recommendation. http://www.w3.org/TR/xmlschema-0/

[XML01b] BIRON, Paul V. & MALHOTRA, Ashok (Hrsg.): *XML Schema Part 2: Datatypes*. W3C. 02.05.2001. W3C Recommendation. http://www.w3.org/TR/xmlschema-2/

[XPa99] CLARK, James & DEROSE, Steve (Hrsg.): *XML Path Language (XPath) Version 1.0*. W3C. 16.11.1999. W3C Recommendation. http://www.w3.org/TR/1999/REC-xpath-19991116

[XPa02] BERGLUND, Anders; BOAG, Scott; CHAMBERLIN, Don; FERNANDEZ, Mary; KAY, Michael; ROBIE, Jonathan & SIMÉON, Jérôme (Hrsg.): *XML Path Language (XPath) Version 2.0*. W3C. 15.11.2002. W3C Working Draft. http://www.w3.org/TR/2001/WD-xpath20-20011220

[XQu01] BOAG, Scott; CHAMBERLIN, Don; FERNANDEZ, Mary F.; FLORESCU, Daniela; ROBIE, Jonathan & SIMÉON, Jérôme (Hrsg.): *XQuery 1.0: An XML Query Language*. W3C. 20.12.2001. W3C Working Draft. http://www.w3.org/TR/xquery/

[XSL99] CLARK, James (Hrsg.): *XSL Transformations (XSLT) Version 1.0*. W3C. 16.11.1999. W3C Recommendation. http://www.w3.org/TR/xslt

[XSL01a] ADLER, Sharon; BERGLUND, Anders; CARUSO, Jeff; DEACH, Stephen; GRAHAM, Tony; GROSSO, Paul; GUTENTAG, Eduardo; MILOWSKI, Alex; PARNELL, Scott; RICHMAN, Jeremy & ZILLES, Steve (Hrsg.): *Extensible Stylesheet Language (XSL) Version 1.0*. W3C. 15.10.2001. W3C Recommendation. http://www.w3.org/TR/xsl/

[XSL01b] KAY, Michael (Hrsg.): *XSL Transformations (XSLT) Version 2.0*. W3C. 20.12.2001. W3C Working Draft. http://www.w3.org/TR/2001/WD-xslt20-20011220/

[XTM01] PEPPER, Steve & MOORE, Graham (Hrsg.): *XML Topic Maps (XTM) 1.0*. TopicMaps.Org. 06.08.2001. TopicMaps.Org Specification. http://www.topicmaps.org/xtm/1.0/

[Zsc00] ZSCHAU, Oliver: *Web Content Management. Einführung – Konzepte – Software*, University of Applied Sciences Mittweida (FH), Diplomarbeit, 2000